Revision-Based Generation of Natural Language Summaries Providing Historical Background

Corpus-Based Analysis, Design, Implementation and Evaluation

Jacques Robin

Technical Report CUCS-034-94

Submitted in partial fulfillment of the
requirements for the degree
of Doctor of Philosophy
in the Graduate School of Arts and Sciences

Columbia University

December 1994

Abstract

Revision-Based Generation of Natural Language Summaries
Providing Historical Background:
Corpus-Based Analysis, Design, Implementation and Evaluation
Jacques Robin

Automatically summarizing vast amounts of on-line quantitative data with a short natural language paragraph has a wide range of real-world applications. However, this specific task raises a number of difficult issues that are quite distinct from the generic task of language generation: conciseness, complex sentences, floating concepts, historical background, paraphrasing power and implicit content.

In this thesis, I address these specific issues by proposing a new generation model in which a first pass builds a draft containing only the essential new facts to report and a second pass incrementally revises this draft to *opportunistically* add as many background facts as can fit within the space limit. This model requires a new type of linguistic knowledge: revision operations, which specifyies the various ways a draft can be transformed in order to concisely accommodate a new piece of information. I present an in-depth corpus analysis of human-written sports summaries that resulted in an extensive set of such revision operations. I also present the implementation, based on functional unification grammars, of the system STREAK, which relies on these operations to incrementally generate complex sentences summarizing basketball games. This thesis also contains two quantitative evaluations. The first shows that the new revision-based generation model is far more robust than the one-pass model of previous generators. The second evaluation demonstrates that the revision operations acquired during the corpus analysis and implemented in STREAK are, for the most part, portable to at least one other quantitative domain (the stock market).

STREAK is the first report generator that systematically places the facts which it summarizes in their historical perspective. It is more concise than previous systems thanks to its ability to generate more complex sentences and to opportunistically convey facts by adding a few words to carefully chosen draft constituents. The revision operations on which STREAK is based constitute the first set of corpus-based linguistic knowledge geared towards incremental generation. The evaluation presented in this thesis is also the first attempt to quantitatively assess the robustness of a new generation model and the portability of a new type of linguistic knowledge.

Contents

1 Introduction **1**

1.1 Starting points . . . 1

1.1.1 The need for summary report generation . . . 1

1.1.2 Summarization issues . . . 2

1.2 Aspects of the research . . . 7

1.2.1 An opportunistic generation model . . . 7

1.2.2 A corpus analysis to acquire revision rules . . . 10

1.2.3 An implemented revision-based generator . . . 12

1.2.4 A quantitative evaluation of the opportunistic model and revision rules . . . 13

1.3 Contributions . . . 14

1.4 Guide to the rest of the thesis . . . 15

2 Corpus analysis of newswire reports **16**

2.1 Motivation and methodology . . . 16

2.2 Focusing on a sub-corpus . . . 17

2.2.1 Report structure and discursive focus . . . 17

2.2.2 Content types and semantic focus . . . 17

2.3 Domain ontology . . . 20

2.4 Corpus sentence structures . . . 21

2.4.1 Top-level structure: concept clustering . . . 21

2.4.2 Internal cluster structure: realization patterns . . . 24

2.5 Revision tools . . . 26

2.5.1 Differential analysis of realization patterns: identifying revision tools . . . 26

2.5.2 Classifying revision operations . . . 27

2.6 Summary . . . 32

3 A new generation architecture **35**

3.1 Text generation subtasks . . . 35

3.2 Motivation for a new architecture . . . 37

3.2.1 From corpus observations to needed abilities . . . 37

3.2.2 From needed abilities to design principles . . . 41
3.3 A revision-based architecture for incremental generation . . . 43
3.3.1 Internal utterance representation . . . 43
3.3.2 Processing components and overall control . . . 54
3.4 Generation subtasks revisited . . . 58
3.5 Summary . . . 59

4 Implementation prototype: the STREAK generator 60

4.1 Goals and scope of the implementation . . . 60
4.2 The internal representation languages . . . 61
4.2.1 The DSS language . . . 62
4.2.2 The SSS language . . . 70
4.2.3 The draft representation . . . 74
4.3 The processing components . . . 76
4.3.1 The phrase planner . . . 76
4.3.2 The lexicalizer . . . 81
4.3.3 The reviser . . . 84
4.4 STREAK at work: two example runs . . . 91
4.4.1 Chaining revisions to generate a complex sentence from a simple draft . . . 91
4.4.2 Choice of revision rule constrained by the surface form of the draft . . . 95
4.5 STREAK by the numbers . . . 96

5 Evaluation 99

5.1 Evaluation and language generation . . . 99
5.1.1 The difficulty of the issue . . . 99
5.1.2 The rising interest in the issue . . . 101
5.1.3 The diversity of the issue . . . 102
5.1.4 The evaluation approach of this thesis . . . 104
5.2 Quantitative evaluation of robustness . . . 106
5.2.1 Review of acquisition corpus results . . . 106
5.2.2 Evaluation goals and test corpora . . . 107
5.2.3 Evaluation parameters . . . 107
5.2.4 Partially automating the evaluation . . . 115
5.2.5 Results . . . 117
5.3 Quantitative evaluation of portability . . . 121
5.3.1 Starting point . . . 121
5.3.2 Methodology . . . 122
5.3.3 Results . . . 129

6 Related work **140**
6.1 Related work in summary report generation . . . 140
6.1.1 Ana . . . 140
6.1.2 Systems based on the Meaning-Text Theory . . . 141
6.1.3 Other summary report generators . . . 146
6.1.4 Comparison with STREAK . . . 146
6.2 Related work in revision-based generation . . . 149
6.2.1 Yh . . . 149
6.2.2 weiveR . . . 150
6.2.3 At what level to perform revision? . . . 150
6.2.4 Revising to satisfy what goals? . . . 152
6.3 Related work in incremental generation . . . 152
6.3.1 Incremental generation and Tree Adjoining Grammars . . . 152
6.3.2 Incremental generation in IPF . . . 160
6.4 Related work in evaluation for generation . . . 161

7 Conclusion **163**
7.1 Contributions . . . 163
7.1.1 Contributions to summary report generation . . . 163
7.1.2 Contributions to generation architecture . . . 164
7.1.3 Contributions to revision-based generation and incremental generation . . . 165
7.1.4 Contributions to evaluation in generation . . . 165
7.1.5 Contributions to SURGE . . . 165
7.2 Limitations and future work . . . 166
7.2.1 Limitations linked to the use of FUF . . . 166
7.2.2 Limitations proper to STREAK . . . 170
7.2.3 Implementing fact generation and discourse planning . . . 172
7.2.4 Further evaluations . . . 174

A A Complete Inventory of Revision Operations **176**
A.1 Monotonic revisions . . . 177
A.1.1 Adjoin . . . 177
A.1.2 Absorb . . . 182
A.1.3 Conjoin . . . 184
A.1.4 Append . . . 188
A.2 Non-monotonic revisions . . . 189
A.2.1 Recast . . . 189
A.2.2 Adjunctization . . . 191
A.2.3 Nominalization . . . 195

A.2.4 Demotion 196
A.2.5 Coordination Promotion 198
A.3 Side transformations 200
A.3.1 Reference adjustment 200
A.3.2 Argument control 201
A.3.3 Ellipsis 202
A.3.4 Scope marking 202
A.3.5 Ordering adjustment 203
A.3.6 Lexical adjustment 203

B Implementation environment: the FUF/SURGE package 204
B.1 The FUF programming language 205
B.1.1 Functional Descriptions 205
B.1.2 Paths 207
B.1.3 Functional Grammars 207
B.2 The SURGE-1.0 unification grammar of English 220
B.2.1 The task of a syntactic grammar 220
B.2.2 The clause sub-grammar 223
B.2.3 The nominal sub-grammar 231
B.2.4 Rest of the syntactic grammar 233
B.3 SURGE-2.0: extensions to complex sentences and quantitative nominals 236
B.3.1 Starting point 236
B.3.2 Extensions to the clause sub-grammar 238
B.3.3 Extensions to the nominal sub-grammar 250
B.3.4 Extensions to the rest of the grammar 250
B.3.5 Summary 250
B.4 Limitations of SURGE 2.0 252
B.4.1 Transitivity 252
B.4.2 Adverbials 252
B.4.3 Mood 253
B.4.4 Voice 253
B.4.5 Nominals 253
B.5 Coverage of SURGE-2.0 254
B.5.1 Test inputs for the extensions at the nominal rank 254
B.5.2 Test inputs for the extensions at the complex sentence rank 282

C Example runs of STREAK 308
C.1 Original example runs with input and draft representation 308
C.1.1 Example run 1: Chaining different revision rules to generate a complex sentence . . . 308

C.1.2 Example run 2: Choice of revision rule constrained by the surface form of the draft . . 327
C.2 Additional example runs 335
C.2.1 Example run 3: Generating draft paraphrases 335
C.2.2 Example run 4: Using revision to generate paraphrases 337
C.2.3 Example run 5: Applying the same revision rules in different contexts 339

D Partially automating corpus analysis: the CREP software tool 341

D.1 A brief overview of CREP syntax 341
D.2 Using CREP for lexical knowledge acquisition 342
D.3 Encoding realization patterns as CREP expressions 343
D.4 Porting the signature of a revision tool: a detailed example 347

List of Figures

1.1 A human-written newswire summary 4
1.2 The report of Fig. 1.1 stripped off its historical information. 4
1.3 The facts in the lead sentence of the report in Fig. 1.1 4
1.4 Paragraph made of simple sentences, paraphrasing the complex lead sentence of Fig. 1.1 . . . 4
1.5 Variety of syntactic paraphrases 5
1.6 Traditional sentence generation model 8
1.7 Opportunistic sentence generation model 8
1.8 Paired corpus sentences revealing revision operations 11

2.1 Standard box-score and corresponding natural language report 19
2.2 Domain ontology 20
2.3 Top-level structures of basic sentences 23
2.4 Complex sentence with top-level hypotactic structure 23
2.5 Concept clusters 24
2.6 Realization pattern examples 25
2.7 Differential analysis of realization patterns 27
2.8 Revision operation hierarchy: top-levels 28
2.9 `Adjoin` schema 28
2.10 `Adjoin` revision operation hierarchy 29
2.11 `Adjunctization` schema 30
2.12 Adjunctization revision operation hierarchy 31

3.1 Syntactic diversity of historical information 38
3.2 Streak information in a corpus report 40
3.3 Internal representation layers 44
3.4 An example DSS: `dss0` 45
3.5 `sss1` (left) and `sss2` (right): two different perspectives on `dss0` 48
3.6 Content units of `dss0` explicitly realized in `sss1` 48
3.7 Content units of `dss0` explicitly realized in `sss2` and `sss3` 48
3.8 `sss3`: a third perspective on `dss0` including a rhetorical event 50
3.9 `dgs21` (left) and `dgs22` (right): two different lexicalizations of `sss2` 50

3.10 dgs1: an example lexicalization of sss1 . . . 50
3.11 dgs3: an example lexicalization of sss3 . . . 51
3.12 An example of three-layered draft representation . . . 52
3.13 A revision-based generation architecture: overall picture . . . 55
3.14 A revision-based generation architecture: **draft time** . . . 56
3.15 A revision-based generation architecture: **revision time** . . . 57

4.1 A very simple conceptual network . . . 62
4.2 Representing a flat conceptual net as an FD . . . 63
4.3 Example of fact in input format in STREAK . . . 64
4.4 Example of input FD for basic draft . . . 66
4.5 Example of conceptual network containing an historical record fact . . . 68
4.6 FD representation of the subnet of Fig. 4.5 . . . 69
4.7 FD representing a streak extension . . . 71
4.8 FD representing a streak record update . . . 72
4.9 FD encoding of sss1 . . . 75
4.10 FD encoding of sss2 . . . 75
4.11 FD encoding of sss3 . . . 76
4.12 Example FD encoding a paratactically structured SSS . . . 77
4.13 FD representing the draft phrase "the Orlando Magic defeated the Toronto Raptors" . . . 78
4.14 Partial SSS after top-level unification . . . 79
4.15 Example of phrase planning rules . . . 80
4.16 Encoding verb choice in STREAK . . . 83
4.17 Encoding the nominalization revision tool in STREAK . . . 86
4.18 Testing game result material clauses and losing streak extensions . . . 87
4.19 STREAK generating a simple draft and incrementally revising it into a complex one . . . 93
4.20 STREAK using different revision rules depending on the surface form of the draft . . . 97

5.1 Test corpus partitions for conceptual and clustering coverage . . . 109
5.2 Test corpus partitions for paraphrasing coverage . . . 111
5.3 Signature of Adjoin of Classifier w/ example phrases . . . 122
5.4 Example of non-overlapping set of target clusters for a given tool . . . 123
5.5 CREP expressions approximating the signature of Adjoin of Relative Clause to Top NP . 124
5.6 Definition file for the sub-expressions of Fig. 5.5 . . . 125
5.7 Revision operation hierarchy: top-levels . . . 134
5.8 Adjoin revision operation hierarchy . . . 134
5.9 Conjoin revision operation hierarchy . . . 134
5.10 Absorb revision operation hierarchy . . . 135
5.11 Adjunctization revision operation hierarchy . . . 135

5.12 Recast and Nominalize revision operation hierarchy 135
5.13 Demotion and Promotion revision operation hierarchy 135

6.1 An example report generated by Ana (from [Kukich 1983]) 141
6.2 Example fact and message in ANA (from [Kukich 1983]) 142
6.3 Example phrasal lexicon entry in ANA (from [Kukich 1983]) 142
6.4 An example report generated by FOG (from [Polguere 1990]) 143
6.5 An example report generated by GOSSIP (from [Carcagno and Iordanskaja 1993]) 143
6.6 Conceptual Communicative *vs.* Deep Semantic representations the Meaning-Text Theory (from [Carcagno and Iordanskaja 1993]) 144
6.7 Example text generated by YH: initial and final drafts (from [Gabriel 1988]) 149
6.8 Adjoining in TAGs 153
6.9 Substitution of minimal trees in lexicalized TAGs 154
6.10 Mapping rules in synchronous TAGs 157
6.11 Example input/output of a systemic TAG generator 158

7.1 Canonic and non-canonic list representations of the same FD 167
7.2 Unique graph corresponding to the lists of Fig. 7.1 168

A.1 Revision operation hierarchy: top-levels 176
A.2 **Adjoin** schema 178
A.3 Adjoin revision operation hierarchy 179
A.4 **Absorb** schema 183
A.5 Absorb revision operation hierarchy 183
A.6 **Conjoin** schema 185
A.7 Conjoin revision operation hierarchy 186
A.8 **Append** schema 188
A.9 **Recast** schema 190
A.10 Recast revision operation hierarchy 190
A.11 **Adjunctization** schema 192
A.12 Adjunctization revision operation hierarchy 193
A.13 **Nominalization** schema 195
A.14 Append and Nominalization revision operation hierarchy 196
A.15 **Demotion** schema 197
A.16 Demotion and Promotion revision operation hierarchy 198
A.17 **Promotion** schema 199

B.1 Space of FDs 206
B.2 FD as equation set 207
B.3 Simple backtracking example 208

B.4 Constituent recursion . . . 209
B.5 Relative *vs.* absolute paths . . . 211
B.6 The FD of figures 1.2 and 1.5 viewed as a graph . . . 211
B.7 Unidirectional *vs.* bidirectional unification . . . 212
B.8 Using `control` to encode negation . . . 213
B.9 Modular definition of an FG . . . 213
B.10 Insert-fd: pasting a smaller FD into a larger FD . . . 215
B.11 Relocate: cutting a smaller FD from a larger FD . . . 216
B.12 Alternative list representations of the same FD . . . 218
B.13 Enriched specification for the sentence "They are winning" . . . 221
B.14 Simple process hierarchy in SURGE . . . 226
B.15 Composite processes in SURGE . . . 228
B.16 Circumstantials in SURGE-1.0 . . . 229
B.17 Mood in SURGE-1.0 . . . 230
B.18 Nominal-Internal grammatical functions and fillers in SURGE-1.0. . . . 232
B.19 Example input descriptions of paratactic complexes in SURGE . . . 234
B.20 Nuclearity of clause constituents . . . 240
B.21 Thematic to syntactic role mapping for adverbials in SURGE-2.0 . . . 241
B.22 Predicate Adjuncts in SURGE-2.0 . . . 243
B.23 Sentence Adjuncts in SURGE-2.0 . . . 244
B.24 Disjuncts in SURGE-2.0 . . . 245
B.25 Extensions to the mood system . . . 249
B.26 Example input description with controlled subject . . . 249
B.27 Extensions to the nominal grammar . . . 251
B.28 Input description of a complex nominal in SURGE-2.0 . . . 251

C.1 Applying a different rule at each revision increment . . . 309
C.2 Form flags used in the example runs . . . 311
C.3 Input containing the four fixed facts for Draft 0 in *Run 1* . . . 312
C.4 List of floating fact lists in *Run 1*, with first sub-element . . . 313
C.5 Second floating fact in *Run 1* . . . 314
C.6 Third and fourth floating facts in *Run 1* . . . 315
C.7 Fifth floating fact in *Run 1* . . . 316
C.8 Sixth, seventh and eigth floating facts in *Run 1* . . . 317
C.9 DSS layer of final draft in *Run 1*. Continued next page . . . 318
C.9 DSS layer of final draft in *Run 1*. Continued from previous page and continued next page . . 319
C.9 DSS layer of final draft in *Run 1*. Continued from previous page and continued next page . . 320
C.9 DSS layer of final draft in *Run 1*. Continued from previous page . . . 321
C.10 SSS layer of final draft in *Run 1*. Continued next page . . . 322

C.10 SSS layer of final draft in *Run 1.* Continued from previous page 323
C.11 DGS layer of final draft in *Run 1.* Continued next page 324
C.11 DGS layer of final draft in *Run 1.* Continued from previous page 325
C.12 STREAK using different revision rules depending on the surface form of the draft 328
C.13 Pre-conditions to using nominalization for losing streak extensions 329
C.14 DSS layer of both drafts in *Run 2* 330
C.15 SSS layer of the first initial draft in *Run 2* 331
C.16 DGS layer of the first initial draft in *Run 2* 332
C.17 SSS layer of the second initial draft in *Run 2* 333
C.18 DGS layer of the second initial draft in *Run 2* 334
C.19 STREAK generating draft paraphrases 336
C.20 STREAK generating paraphrases by using a variety of revision rules 338
C.21 Complex sentence generation where the same revision rule is used at different draft levels . . 340

D.1 CREP search result for rebounding statistic verbs 343
D.2 Realization pattern examples (reproduced from Section 2.4.2) 345
D.3 CREP expression E2 for realization pattern R2 345
D.4 CREP sub-expression definitions for realization pattern *R*2 346
D.5 Original signature of `Adjoin of Co-Event Clause to Clause` in sports domain 348
D.6 Signature of `Adjoin of Co-Event Clause to Clause` ported to the financial domain 349

Acknowledgements

NEW YORK (CPI[1]) – Jacques Robin wrote a career high 357 pages, published 10 papers and programmed over 200K of FUF and LISP code to pursue his personal turnaround and lead the French Generation to a quadruple overtime victory over the PhD. Dissertations, that allows the New York based team to threepeat under the tenure of Coach Kathy McKeown.

Robin's record performance was made possible by the outstanding commitment to defense of John Kender, Karen Kukich, Steve Feiner and Diane Litman, the latter two repeating their flawless performance of the last series in which they played a key role in the French's triumph over the Thesis Proposals. Kukich also tremendously helped at the offensive end with her contagious enthusiasm and her judiciously PLANned Dunks which came at crucial moments, triggering thunderous applause that further demoralized the opposition.

The prevalent wisdom among forecasters during the pre-season was that this year, the Generation would struggle even to barely survive the playoff qualifiers. After losing Franky "Mr. Efficiency" Smadja, the all-time leader in the papers per minutes played category, to an early retirement two seasons ago, team captain Michael "The Guru" Elhadad just stunned the world by accepting a deep salary cut to sign as a free-agent with the Negev Meta-Barons the very summer after he became the first player ever to rank in the top 5 in 10 distinct statistical categories (research insights, philosophical foundations, linguistic erudition, lisp wizardry, emacs macro hacking, pedagogical excellence, consulting expertise, business acumen, familial nurturing and religious commitment).

Coach McKeown's repeated early warning that her team could threepeat even without Mike, was initially dismissed by many as little more than her latest rhetorical schema to keep her group confident and under pressure. Once more she proved all her critics wrong, becoming the first coach to ever mount a successful title defense without the Most Valuable Phd from the previous year for the second straight time. Though her famous strategical approach, copied all over the league, has always been highly rated, it is only this year that she finally gets all due credit for her scouting prowess.

Who else could have masterminded the acquisition of Eric "Data" Siegel, who is rapidly evolving as a key player for the Tetra, while his fitness measure, under the constant prodding of team aerobic conditioner Tony Weida, sets an example for the rest of the Generation? Or dared to bet on Vasileios "Hatzi" Vassiloglou, whom she snatched from the Carnegie Mellons for a second round pick when he was little more than an ex Greek League scoring champion with a smoking problem, and predicted that, staying for longer late night practices than anyone else, it would take him only a few months to become so statistically significant as to inspire nothing but fear in the whole Eastern Conference.

But beyond shrewd recruiting, it is perhaps the making of a late-blooming and unlikely all-star that will remain as Coach McKeown's most remarkable legacy this year. The turning point came when the Generation were riding a four week losing streak and she decided to insert Robin in the starting line-up. This bold move surprised everybody, not the least Robin himself. "It was really the last thing I expected" he explains. "I was worn out by the grueling road trip that we had just completed, which included back to back games against the Qualifying Examinations, the Teaching Assistantships, the Teaching Requirements and the Area Papers, and I was really discouraged by the little research minutes I was seeing off the bench. So when I entered Coach McKeown's office that day, I just told her that I really felt that I did not have what it takes to ever become an impact player and thus preferred to quit. Leaving her office a starter after such defeatist discourse really felt like an hallucination. In retrospect though, her improbable reaction is a perfect example of her unique ability to pragmatically make the most of any situation."

Head Coach McKeown was not the only person Robin wanted to thank in his post title defense game interview. "I would also like to thank Assistant Coaches Becky Passoneau and Judith Klavans for their tactical advice, teammate Darrin "Plaster Cramp" Duford for assisting on so many of my points and weight trainer Mel Francis for keeping me strong. Together with Israel and Tamar Ben-Shaul, Doree Seligmann, Tom O'Donnell, Paul Michelman, Pino Italiano and Pascale Fung, they also helped remind me that even under the harrowing pressure of the professional level where results are everything, one should still try to have some fun along the way. Thanks also to my agents Danilo and Patricia Florissi and Sandra Aluisio

[1] CyberPress International

for getting me the best possible trade to Brazil. On the motivational side, I must cite the ultimate warrior, Bülent “Phoenix” Yener for leading by example with his survival camp mentality and almost winning me over to the idea that one should never ever give up, even in the face of overwhelming and seemingly random cruel adversity. In that respect, I am also indebted to Pierre Moroni for providing the initial confidence boost. Finally on the personal side, I like would to keep my very most heartfelt thanks to Thierno Mol, Jim Cardoverde, Terremotto and Dynamo for their invaluable complicity and above all to yèné fëǩër Nunu Challa and my parents Jean and Jacqueline for their patient, unabated, unconditional and all around support.”

A la mémoire chérie de Lulu ...

Chapter 1

Introduction

1.1 Starting points

1.1.1 The need for summary report generation

In recent years, the volume of information available on-line has grown exponentially, and several large-scale efforts, such as the information superhighway and digital library initiatives, are currently under way to further accelerate this growth. In order to put this abundance of electronic data to good use and avoid information inundation, the development of automatic summarization facilities is critical.

Three fields of computer science can contribute to the development of such facilities: information retrieval, multi-media presentations and natural language processing. Concerning the use of natural language, there is an important distinction between two forms of on-line information: textual form whose content is unrestricted and tabular form whose content is only quantitative. Summarizing free text generally requires unrestricted natural language interpretation, which has remained too elusive a research goal for immediate application. In contrast, automatically producing a natural language paragraph summarizing a large amount of quantitative data - unexploitable in its original exhaustive and tabular form - is possible today. In fact, in a recent round-table on technology transfer [Zock *et al.* 1992], several participants singled out this particular task as the most promising industrial application of natural language generation. Meteorological measurements, stock market indexes, financial audit data, computer surveillance trails, labor and census statistics, sports statistics and hospital patient histories are but a few examples illustrating the pervasiveness of on-line data that is best exploited when summarized by a short natural language text. Indeed, one of the only two language generation systems currently in daily use[1], produces bilingual weather forecasts for the Canadian weather service[2].

In this thesis, I show that generating summaries raises a number of difficult issues that are quite distinct from the generic task of language generation. To address these specific issues, I propose a new generation model where a first pass builds a draft containing only the essential new facts to report and a second pass incrementally revises this draft to *opportunistically* add as many background facts as can fit within the space limit. This model requires a new type of linguistic knowledge: revision operations, specifying the various ways a draft can be transformed in order to concisely accommodate a new piece of information. I present both an in-depth corpus analysis of human-written sports summaries that resulted in an extensive set of such revision operations, and the implementation of the system STREAK[3] which relies on these operations to incrementally generate complex sentences summarizing basketball games. I also quantitatively evaluate the advantages of this new revision-based generation model over the one-pass model of previous generators

[1] cf. FOG [Bourbeau *et al.* 1990], see Section 6.1 for details.

[2] The other system working as a real-world application is PLANDOC [Kukich *et al.* 1994] [McKeown *et al.* 1995], which produces automated documentation for managers about the choices of telephone network planning engineers at Bellcore.

[3] Surface Text Reviser Expressing Additional Knowledge.

in terms of same-domain robustness and cross-domain portability. This evaluation used test corpora in both the sports and financial domains, where test data was distinct from the acquisition data.

1.1.2 Summarization issues

By inspecting a corpus of sports reports[4] like the one in Fig. 1.1, I identified seven challenging issues that summarization raises for language generation systems. Most of these issues have been largely ignored by previous generators which did not work within space limits. These issues include (1) floating concepts, (2) historical background, (3) conciseness, (4) sentence complexity, (5) paraphrasing, (6) granularity and (7) implicit content. The reports of the corpus I analyzed summarize basketball games. In each of these reports, the lead sentence itself summarizes the rest of the report [Mencher 1984] [Fensch 1988].

1.1.2.1 Floating concepts

While some concepts consistently appear in *fixed* locations across reports (*e.g.*, the final score of a ballgame is always conveyed in the second half of the lead sentence), others *float*, appearing potentially anywhere in the report structure. Consider for example the two instances of the streak concept[5] in Fig. 1.1. One instance (boldfaced on line 2), concerning the Nuggets, is conveyed in the lead. But the other, concerning the Kings, is the very last fact conveyed in the report. Floating concepts thus appear to be *opportunistically* realized where the form of the surrounding text allows. Consider the Kings' losing streak in Fig. 1.1. Instead of the closing sentence, it could have alternatively been attached to either the lead sentence or the second sentence, since both of them also contain a reference (underlined) to the Kings. The particular choice of the closing sentence seems to be motivated by stylistic surface form factors: the lead sentence is already complex enough and the reference to the Kings' in the second sentence is too deeply embedded to be the object of an apposition. Similar considerations probably explain the choice of a separate sentence, the third in the report, as opposed to attachment to the lead, for the remaining facts concerning the Nuggets.

The flexibility with which floating facts can be conveyed is an asset for a summarization application: among the various forms and locations possible for their inclusion in the report, the most concise one can be chosen. However, for a generator this flexibility represents a difficulty: it requires searching a much larger space of expressive options. The basic idea underlying the draft and revision approach proposed in this thesis is to use the inherent rigidity of the fixed facts as constraints to reduce the search space of options for the expression of the floating ones.

1.1.2.2 Historical background

Although optional in any given sentence, floating concepts cannot be ignored: in the corpus I analyzed, they account for over 40% of total lead sentence content. But what makes floating concepts even more crucial than this pervasiveness is that they cover entire content types. One such type is historical information (*i.e.,* past facts related to the new reported ones which explain their relevance). The importance of such background facts is illustrated in Fig. 1.1, where they are boldfaced. Omitting them would result in a much impoverished report as shown in Fig. 1.2, where the same report has been stripped from these historical facts. In the corpus I analyzed, 65% of the lead sentences contained some historical fact. Yet, previous summary report generation systems were not capable of including such background information in their reports due to its floating nature. Conveying such facts requires a generator to access an historical database to supplement the table of new statistics it receives as input. But it allows the generator to *contextualize* its input data in addition to *summarizing* it. Readers do not merely want to know what happened but also why it is interesting.

[4] Taken from the UPI newswire.

[5] *i.e.,* information about series of similar outcome.

1.1.2.3 Conciseness

A generator can perform two different types of summarization: *conceptual summarization* by selecting the essential facts, and *linguistic summarization* by expressing the selected facts in compact surface form. Considering the historical background dramatically increases the number of candidate facts to include in the report and thus emphasizes the need for linguistic summarization. When limited space is available, background facts cannot be conveyed by separate sentences. Instead, as in Fig. 1.1, they can be concisely expressed by small phrases, sometimes down to a single word (*e.g., "rookie"* in front of the last sentence), woven into the expression of the new facts which provide the report structure backbone. This is always possible, since an historical fact is relevant only if it is related to some new fact to report. Previous report generators focused either on performing conceptual summarization and/or on a form of linguistic summarization limited to clause combining and anaphora. They therefore failed to exploit the full potential for conciseness laying inside each clause.

1.1.2.4 Sentence complexity

A major strategy to achieve conciseness is to use complex sentences[6]. This is why sentences in newswire reports tend to be very long, especially lead sentences which themselves summarize the rest of the report by packing together all the crucial facts. For example, in Fig. 1.1 the lead sentence alone conveys ten facts. An intuitive, logical form representing these facts is given in Fig. 1.3.

The compactness of complex sentences is illustrated in Fig. 1.4. It contains a multi-sentence paragraph that paraphrases the lead sentence of the report of Fig. 1.1. In this paragraph, each of the ten facts packed inside this lead sentence is conveyed by an independent, simple sentence. This report is an example of what a system performing only conceptual summarization and a form of linguistic summarization limited to anaphora can generate. In number of words, it is twice the length (78 *vs.* 36) of the synonymous complex sentence. Complex sentences are concise because, by grouping together several facts, they can factor out their common content units. For example, there are five occurrences of the unit `(michael,adams)` in the ten facts of Fig. 1.3. Grouping these ten facts in the complex lead of Fig. 1.1 allow collapsing these five occurrences in a single referring NP.

The table below contrasts the complexity of two previous summary report generators[7], GOSSIP [Carcagno and Iordanskaja 1993] and ANA [Kukich 1983], with the complexity of the lead sentences in the corpus I analyzed. Its rows indicate the number of facts, represented in logical form as in Fig. 1.3, parse tree depth and number of words[8] of a sentence. For GOSSIP and ANA, these numbers are based on the few example reports provided in publications about these systems[9].

	Previous systems		Human-written lead sentences	
	GOSSIP	ANA		
	max	max	min	max
Factual density	5	4	4	12
Syntactic depth	5	6	4	10
Lexical length	17	34	23	46

This table points out a complexity gap between sentences generated by these systems and the ones observed in human-written newswire summaries. One reason why these two systems could produce good summaries in their respective domains with simpler sentences is that they excluded references to historical facts from their target sublanguage. Other systems that generate reports that are not specifically summaries, can avoid generating very complex sentences since they are under no pressure to be concise. Two such systems, Danlos' generator [Danlos 1986] and PAULINE [Hovy 1988], generate sentences of a complexity

[6] As paradoxical as it may first sound.
[7] cf. Section 6.1 for details on these two systems.
[8] Counting both open and closed class words.
[9] Though by no mean a rigorous, systematic comparison, it nonetheless gives an estimate that is fair enough for the point I make here.

Sacramento, Ca. – Michael Adams scored **a career-high** 44 points Wednesday night, including seven 3-point baskets, to help the short-handed Denver Nuggets **end a five-game losing streak** with a 128-112 victory over the Sacramento Kings.
Adams, **who was drafted and then discarded by the Kings four seasons ago**, made 17 of 26 field goals, including seven of 11 3-point attempts, and hit three of four free throws **to break his previous career high of 35 points**.
The Nuggets, who had only eight players available for the game, **improved to 2-12 on the road and 6-20 overall**.
Rookie guard Travis Mays, **playing his second game after missing 11 with back spasms**, scored **a season-high** 36 points for the Kings, **losers of four in a row**.

Figure 1.1: A human-written newswire summary

Sacramento, Ca. – Michael Adams scored 44 points Wednesday night, including seven 3-point baskets, to help the short-handed Denver Nuggets to a 128-112 victory over the Sacramento Kings.
Adams made 17 of 26 field goals, including seven of 11 3-point attempts, and hit three of four free throws.
The Nuggets had only eight players available for the game. Guard Travis Mays scored 36 points for the Kings.

Figure 1.2: The report of Fig. 1.1 stripped off its historical information.

```
F1  = location(sacramento,CA)
F2  = scoring((michael,adams),(44,pt))
F3  = new-high(F2,scoring((michael,adams),(X,pt),lifetime((michael,adams)))
F4  = scoring((michael,adams),(7,3-pointer))
F5  = miss-players(denver,nuggets)
F6  = team((michael,adams),(sacramento,kings))
F7  = beat((denver,nuggets),(sacramento,kings))
F8  = streak-end(F7,(denver,nuggets),5,loss)
F9  = score(128,112)
F10 = time(Wed,night)
```

Figure 1.3: The facts in the lead sentence of the report in Fig. 1.1

Sacramento, Ca. – Michael Adams scored 44 points. This scoring performance was the best of his entire career. It included seven 3-point baskets. Michael Adams plays for the Denver Nuggets. This team was missing many players for the game. But with the performance of Michael Adams, Denver managed to defeat the Sacramento Kings. The Nuggets had lost five consecutive times before this game. The final score was 128-112. The game was played Wednesday night.

Figure 1.4: Paragraph made of simple sentences, paraphrasing the complex lead sentence of Fig. 1.1

1. `Coordinated clause:`
   David Robinson scored 32 points Friday night *lifting the San Antonio Spurs to a 127 111 victory over Denver* **and handing the Nuggets their seventh straight loss**
2. `Qualifying non-finite clause in top-level nominal:`
   David Robinson scored 32 points Friday night lifting the San Antonio Spurs to *a 127 111 victory over Denver* **sending the Nuggets to their seventh straight setback**.
3. `Qualifying relative clause in top-level nominal:`
   David Robinson scored 32 points Friday night lifting the San Antonio Spurs to *a 127 111 victory over the Nuggets* **that extended Denver's losing streak to seven games**.
4. `Qualifying relative clause in embedded nominal:`
   David Robinson scored 32 points Friday night lifting the San Antonio Spurs to a 127 111 victory over *the Denver Nuggets* **who lost for the seventh consecutive time**.
5. `Apposition to top-level nominal:`
   David Robinson scored 32 points Friday night lifting the San Antonio Spurs to *a 127 111 victory over Denver*, **the Nuggets' seventh straight defeat**.
6. `Apposition to embedded nominal:`
   David Robinson scored 32 points Friday night lifting the San Antonio Spurs to a 127 111 victory over *the Denver Nuggets*, **losers of seven in a row**.

Figure 1.5: Variety of syntactic paraphrases

comparable to those of ANA, which constitutes an upper-bound for previous work. However, the compound need to be concise and to convey historical background, requires generating more complex sentences. But planning the content of a single sentence of such complexity becomes at least as difficult as planning the content of an entire paragraph made of simple sentences. This is illustrated by the paragraph in Fig. 1.4, which conveys the same facts as the lead sentence in Fig. 1.1.

Though complex sentences allow concise expression of multiple facts and are thus very informative, beyond a certain point they can also become unreadable. A summary report generator thus needs to carefully monitor sentence complexity to stay within the readability threshold. However, such thresholds can only be defined in terms of surface form factors such as number of words or depth of embedding. Planning a maximally complex sentence such as those observed in newswire summaries can only be done under surface form constraints. The standard content planning techniques devised for paragraphs such as textual schemas [McKeown 1985] and Rhetorical Structure Theory [Hovy 1991] cannot be used for the complex sentences of summaries precisely because they operate purely at the conceptual level.

1.1.2.5 Paraphrasing power

Conveying floating facts concisely requires attaching them opportunistically where the surrounding text allows. These facts thus need to be expressible by a wide variety of syntactic constructs, each suitable to a particular textual context. Even in a fixed context, surface form variation is also needed: one of the surest ways for a generation system to betray its artificial nature is to always output the same linguistic form when given the same type of information in input.

Paraphrasing power is illustrated, for example, in the report of Fig. 1.1, where the same streak concept is expressed by a VP in the lead sentence (boldfaced on line 2) and an NP in the closing one (the very last constituent of the report). In the corpus I analyzed, over 60 different syntactic constructs were used to express this concept. A sample of those constructs is shown in Fig. 1.5, where the same streak concept - highlighted in bold - is conveyed by a different construct in each of six synonymous sentences.

Note that these constructs differ not only in terms of the syntactic category of the constituent conveying the streak (*e.g.,* clause in (4) *vs.* NP in (5)) and the syntactic device linking this constituent to the rest of the sentence (*e.g.,* coordination in (1) *vs.* embedding in (2)), but also in terms of the level at which this device is used inside the sentence structure. In each example, the sentence constituent onto which the streak fact is attached is highlighted in italics: *e.g.,* while in (3) and (5) a relative clause and an appositive nominal are respectively used to modify the NP expressing the whole game result, in (4) and (6) these same two devices are used to modify the NP referring only to the losing team which is embedded deeper in the syntactic structure.

1.1.2.6 Granularity

A crucial feature of a generator is the granularity at which it translates its conceptual input into a linguistic output. The coarse end of the granularity range includes generators that rely on a phrasal lexicon, where entries are entire phrases, each simultaneously expressing several content units. The sentences generated by such systems are thus macrocoded from two to four phrasal entries. At the other end of this range are generators that rely on a word-based lexicon, where entries consists of individual words[10]. The sentences generated by such systems are thus microcoded from words conveying only one or two content units.

GOSSIP and ANA (cf. table of Section 1.1.2.4) illustrate these two extremes. While both generate summaries, the former is representative of the class of microcoded generators that also includes FOG [Bourbeau *et al.* 1990], LFS [Iordanskaja *et al.* 1994], KAMP [Appelt 1985], KALIPSOS [Nogier 1990], FN [Reiter 1991], EPICURE [Dale 1992], IGEN [Rubinoff 1992], COMET [McKeown *et al.* 1990], AVDISORII [Elhadad 1993b] and PLANDOC [Kukich *et al.* 1994] while the latter is representative of the class of macrocoded generators that also includes Danlos' generator [Danlos 1986], PHRED [Jacobs 1985], PAULINE [Hovy 1988], SEMTEX [Roesner 1987] and WEIVER [Inui *et al.* 1992].

The fact that ANA generates more complex sentences than GOSSIP generalizes to the respective classes of generators to which they belong. Thus, existing generators either microcode simpler sentences or macrocode more complex sentences. Generating the kind of sentences of the corpus I analyzed however, requires not only generating even more complex sentences, but also microcoding them. The main reason for this necessity is that microcoding is more compositional and hence makes paraphrasing power easy to scale up with a few additional single word entries. To attain a similar paraphrasing power with macrocoding, a combinatorially explosive number of phrasal entries need to be added to the lexicon. As in the case of choosing where and how to convey floating facts, microcoding a very complex sentence requires exploring a much vaster search space. This is another motivation for the draft and revision model I advocate in this thesis. It decomposes this overall search into a initial draft phase followed by a set of incremental revision steps.

1.1.2.7 Implicit content

Another good strategy for achieving conciseness is to convey content *implicitly.* By implicit content I mean information recoverable though not specifically expressed by any word or linguistic constituent. Such content can be recovered either from the very structure of the sentence (*i.e.*, by how the explicitly expressed content units are organized inside of it) and/or by domain reasoning. Consider again the lead sentence of Fig. 1.1 and the corresponding facts in Fig. 1.1. The fact F_6, that Adams plays for Denver, is not explicitly conveyed in this lead. It is nonetheless immediately recoverable from the use of the verb "to help" linking the main statistic of the game (Adams' performance) to its result (Denver's win) and the domain knowledge that, a player scoring 44 points can only "help" his own team to win the game[11]. The expression of $F6$ in this lead sentence is thus implicit, scattered into the meaning of severals words, each of which independently conveys in itself another content unit. In contrast in the paragraph of Fig. 1.4, $F6$ is explicitly conveyed by

[10] Or short multi-word collocations containing only a single open-class word; for example "*to make up with*" where the verb "*to make*" is the only open-class item would be a single entry, while "*to make a mistake*" would need to be built from the two entries "*to make*" and "*mistake*".

[11] Had the player scored only one point, these two facts could not have been linked by "to help"; "while" or "as" expressing a mere co-occurrence would be needed instead.

a separate sentence. Such implicit content is more frequent in complex sentences packing in many related facts than in simple ones.

This issue influenced the design of the generator STREAK, especially in terms of the internal representation of the draft. Generating implicit content, however, involves many issues of its own and this thesis only scratches the surface[12].

1.2 Aspects of the research

The research presented in this thesis addresses the challenging language generation issues described in the previous section. It has four main aspects:

1. A modelling and design aspect: an opportunistic generation approach.
2. A linguistic knowledge acquisition aspect: a corpus analysis resulting in a set of revision rules to incrementally generate complex sentences from simple ones.
3. An implementation aspect: the STREAK generation system.
4. A quantitative evaluation aspect: measures of robustness and portability.

I survey these aspects in turn in the following subsections.

1.2.1 An opportunistic generation model

The first aspect is the design of a new language generation model. How the overall generation process can be decomposed into modules is still a matter of open debate among researchers in the field. However, the design of most generators built for practical applications essentially varies on a common theme (cf. [Reiter 1994]). Temporarily ignoring the multi-sentential aspect of some generators to focus on sentence generation (I revisit in detail the design issues introduced here within the larger framework of full text generation in Section 3.3.2), this common theme is outlined in Fig. 1.6.

In this traditional sentence generation model, a component interfacing the generator with the underlying application program produces a specification of the content to convey. In the case of summary report generation from quantitative data, this interface is a fact generator that retrieves interesting data from tables of numbers and reformats them as conceptual structures suitable for language generation. For other applications, this interface may query a database, the trace of an expert system, the interlingua representation of a text to translate, etc.

The resulting content specification is passed to a lexicalization component. This component maps the content specification into a linguistic specification of the sentence expressing this content. It chooses the basic syntactic structure of the sentence as well as its open-class lexical items[13]. This linguistic specification is then passed to a syntactic grammar that enforces syntactic rules such as agreement, chooses closed-class words, inflect open-class words and linearize the syntactic structure into a natural language string.

The lexicalizer generally builds the syntactic structure in top-down recursive fashion following the algorithm below:

1. Choose a concept to head the sentence structure.

[12] In particular, I do not discuss its relation to adjacent topics such as user-modeling [Paris 1987], conversational implicatures [Reiter 1991] or argumentation [Elhadad 1993b].

[13] Lexical items are traditionally divided into (a) open-class lexical items (also called cognates) such as nouns, verbs, adjectives and adverbs and (b) closed-class lexical items (also called function words) such as articles, pronouns and conjunctions. Open-classes are large and seemingly ever growing while closed-classes are small and stable. Distinguishing elements in an open-class requires semantics while in a closed-class it can be done on syntactic grounds only. In that sense, prepositions, which are few but cannot be distinguished on purely syntactic grounds (cf. [Herskovits 1986]) are neither really an open nor a closed class of words but a little bit of both.

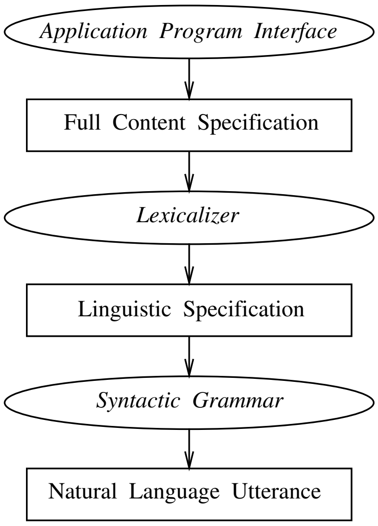

Figure 1.6: Traditional sentence generation model

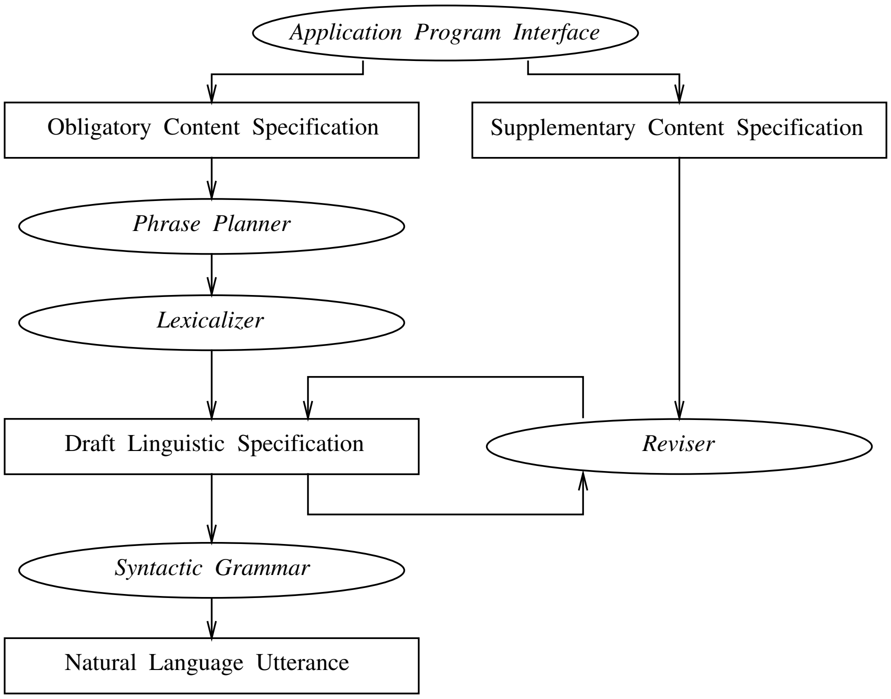

Figure 1.7: Opportunistic sentence generation model

2. Choose a verb lexicalizing this head concept.
3. Choose which other concepts to map onto each argument of this verb.
4. Choose a sentential adjunct for each remaining concept.
5. Recursively choose (a) the syntactic category, (b) head word and (c) internal structure of each verb argument and sentential adjunct.

The number of recursions needed to generate a sentence using this algorithm depends on both the granularity of lexicon (phrasal or word-based) and the complexity of both the content and linguistic specifications.

To illustrate this algorithm with an example, consider generating (with a word-based lexicon) the sentence:
"*Michael Adams scored 44 points Wednesday night*"
from the content specification: `game(scoring((michael,adams),(44,pt)),time(Wed,night))`

For this example, step 1 involves choosing `scoring` as sentence head as opposed to, say, `time` which would result in the alternative output "*The game in which Michael Adams scored 44 points was played Wednesday night*". Step 2 involves choosing the verb "*to score*" instead of, say, "*to have*" or "*to finish with*". Step 3 involves choosing `(michael,adams)` as the agent of "*to score*" and `(44,pt)` as its patient. Step 4 involves mapping `(Wed,night)` onto a time adjunct. To illustrate step 5, consider for instance the recursive treatment of the patient argument: (5a) involves choosing an NP, (5b) choosing `point` as the head of this NP and (5c) choosing to map `44` as a cardinal pre-modifier.

When generating a simple sentence that conveys only fixed facts and for which only a few paraphrases exist, as in the example above, each step in this algorithm involves a few candidates and constraints and is fairly self-contained And without a space limitation, the more problematic floating facts can always be conveyed in a subsequent separate sentence. For example, the sentence above could be followed by "*This scoring performance was the best of his entire career.*"

This is not the case for generating the complex lead sentences of summaries that concisely pack essential fixed facts and related background facts, and for which there are many alternative paraphrases. For such sentences, the steps in the algorithm above would involve far too many candidates and constraints to remain manageable and also become overly interdependent. An example of such complex sentences (where background facts are highlighted in bold) generated by the STREAK prototype is given below:
"Dallas, TX – Charles Barkley **matched his season record with 42 points and Danny Ainge came off the bench to add 21** Friday night as the Phoenix Suns handed the Dallas Mavericks their franchise worst 27th defeat in a row at home 123 - 97."

I propose to generate such complex sentences in two passes. In the first pass, a simpler sentence containing only the fixed facts is generated. This simpler sentence serves as draft material for the second pass, in which floating facts are considered in turn by order of importance. These floating facts are then *opportunistically* woven into the draft through incremental revisions. Where, how and even whether a floating fact is incorporated is constrained by the surface form of the draft.

This new generation model, shown in Fig. 1.7, differs from the traditional model of Fig. 1.6 by three essential properties:

- It decomposes the content to convey into an obligatory part (the fixed facts) and a supplementary part (the floating facts), handling the former prescriptively and the latter opportunistically.
- It decomposes building the linguistic specification into several incremental steps, each one incorporating a supplementary content unit.
- It decomposes building the initial linguistic draft into two distinct processes: phrase planning, where content units are organized inside the sentence structure, and lexicalization, where individual words are chosen for each element in that structure.

The first property means that the generator itself (instead of the underlying application program) has the final say about what supplementary content to actually include in the sentence, allowing linguistic factors to influence content determination. This is essential to pack as many facts as possible in complex sentences while ensuring that they remain readable. The second property multiplies modularity by having the generator taking the steps in the algorithm above at each recursion *and for each increment*. Modularity is also enhanced by the third property, that separates the content organization and the linguistic realization aspects of sentence generation that the traditional algorithm integrates. The phrase planner is in charge of steps 1,3,4 and 5c (content organization) leaving steps 2, 5a and 5b for the lexicalizer (linguistic realization).

This separation is motivated by the two factors below:

- The organization of content can be *lexically* driven only for linguistic constituents at and below the simple clause rank. At the complex sentence rank just as at the paragraph rank, in the absence of a head verb whose argument structure predefines the mapping from content units onto linguistic constituents, this organization can only be *rhetorically* driven.
- In human-written summaries, even the most basic sentences containing only fixed, obligatory facts are multi-clause sentences with several adjuncts. They are thus already at the level of complexity requiring internal rhetorical organization.

Note that there is no feedback from the lexicalizer to the phrase planner. Cases[14] where rhetorical and lexical constraints interact are best handled by the reviser.

Using this model, the complex example sentence given above is generated in the six following steps (the supplementary content unit added at each increment is highlighted in bold):

1. Dallas, TX – Charles Barkley registered 42 points Friday night as the Phoenix Suns routed the Dallas Mavericks 123 - 97.
2. Dallas, TX – Charles Barkley registered 42 points Friday night as the Phoenix Suns **handed** the Dallas Mavericks **their 27th defeat in a row at home** 123 - 97.
3. Dallas, TX – Charles Barkley registered 42 points Friday night as the Phoenix Suns handed the Dallas Mavericks their **franchise worst** 27th defeat in a row at home 123 - 97.
4. Dallas, TX – Charles Barkley **matched his season record with** 42 points Friday night as the Phoenix Suns handed the Dallas Mavericks their franchise worst 27th defeat in a row at home 123 - 97.
5. Dallas, TX – Charles Barkley matched his season record with 42 points **and Danny Ainge added 21** Friday night as the Phoenix Suns handed the Dallas Mavericks their franchise worst 27th defeat in a row at home 123 - 97.
6. Dallas, TX – Charles Barkley matched his season record with 42 points and Danny Ainge **came off the bench to add** 21 Friday night as the Phoenix Suns handed the Dallas Mavericks their franchise worst 27th defeat in a row at home 123 - 97.

From an AI standpoint, this model can be viewed as a heuristic for dealing with the growth of the space of possible linguistic options to search during the generation process in summarization applications. This growth is caused by the compound effects of the necessity to handle (floating) background facts to report new facts in their context, to use complex sentences for their conciseness and to use a word-based lexicon for its scalability.

But this revision-based, opportunistic approach to generation is not only preferable from a system engineering standpoint. An experiment by [Pavard 1985] described in Section 3.2.2 suggests that it is also a more cognitively plausible model for the generation of complex written sentences.

1.2.2 A corpus analysis to acquire revision rules

In order to handle supplementary content opportunistically, the new generation model described above requires the acquisition of a new type of linguistic knowledge structure: revision operations specifying the

[14] Much publicized in the research literature [Appelt 1985] [Danlos 1986] [Meteer 1990] [Rubinoff 1992] yet of marginal significance in most practical applications as pointed out by [Reiter 1994].

Complex sentence S_2 containing two floating facts:
"Houston, TX – Buck Johnson scored **a season high** 26 points Thursday night and the Houston Rockets routed the Orlando Magic 119 95 **for their sixth straight win**".

Sentence S_1 with one less fact than S_2 in the second clause:
"Minneapolis, MN – Pooh Richardson scored **a career high** 35 points Saturday night and the Minnesota Timberwolves beat the Golden State Warriors 121 113."

Sentence S_0 with one less fact than S_1 (and S_1^a) *in the first clause*:
"Chicago, IL – Michael Jordan scored 30 points Saturday night and the Chicago Bulls defeated the Cleveland Cavaliers 108 92."

Sentence S_1^a with one more fact than S_0 but an only partially overlapping syntactic structure:
"Hartford, CT – Karl Malone scored 39 points Friday and the Utah Jazz **handed** the Boston Celtics **their sixth straight home** *loss* 118 94."

Figure 1.8: Paired corpus sentences revealing revision operations

structural transformations that a given draft must undergo to incorporate a given new piece of content. In order to abstract these transformations as well as the semantic, syntactic and lexical constraints that determine their application, I analyzed, down to the individual word, a corpus of lead sentences from basketball game summaries compiled over a season of play. This corpus analysis and the fine-grained linguistic data that resulted from it, constitute the second aspect of the research presented in this thesis.

The main idea driving this corpus analysis was to pair sentences that differ semantically by a single floating fact and identify the minimal syntactic transformation between them. The example corpus sentences in Fig. 1.8 illustrate how I carried out this analysis. In the sublanguage of basketball summaries the fixed facts shared by all sentences are: the most significant statistic by a winning team player, the game result[15], final score-line, location and date. I started from the most complex sentences in the corpus, *i.e.,* those containing several floating facts in addition to the fixed facts. Sentence S_2 is an example of such a sentence. It contains two background facts (in boldface). One appears in the first clause and explains what makes the player statistic conveyed by that clause significant (it is a record). The other appears in the second clause and it relates the game result conveyed by that clause to the winner's previous results (they together form a streak). I then searched the corpus for sentences that lacked one of these two floating facts but otherwise followed a syntactic structure as close as possible to that of S_2. S_1 is an example of such sentence. It differs from S_2 only in that it does not include a streak and lacks the trailing *for*-prepositional phrase. Sentence pair $< S_2, S_1 >$ thus reveals that streaks can be opportunistically added to the report by attaching such *for*-PPs. I then proceeded to search for sentences with one less floating fact than S_1. S_0 is an example of such sentence. It differs from S_1 only in that it does not convey a record and lacks a pre-modifier in front of the statistic NP. Sentence pair $< S_1, S_0 >$ thus reveals that records can be slipped into a report by adding such NP modifiers.

I repeated this pairwise analysis for each distinct sentence structure. It resulted in a set of *revision operations* specifying precise semantic and syntactic constraints on (1) where a particular type of floating fact can be added in a draft and (2) what linguistic constructs can be used for the addition. I classified this set of revision operations in a hierarchy. The top of this hierarchy distinguishes between *monotonic* and *non-monotonic* revisions. The revisions abstracted from the two pairs discussed above are monotonic: the structure of the more complex sentence fully includes that of the simpler one. Such revisions allow the floating fact to be added without disturbing the rest of the sentence. However, the structure of some complex sentences, does not fully overlap with any of the simpler sentence structures conveying one less fact. This is the case for the example of sentence S_1^a in Fig. 1.8. S_0 contains one less fact than S_1^a and has a similar top-level structure and first clause. In S_0 however, the game result is expressed verbally by "to defeat" while

[15] *i.e.,* who won and who lost.

in S_1 it is expressed nominally by "a loss". The revision abstracted from this pair is thus non-monotonic: in order to accommodate the additional streak fact as a nominal modifier (in bold) the expression of the game result has been preallably *nominalized*. This type of non-monotonic revision is one of the most interesting results of the corpus analysis I carried out for this thesis. It empirically confirms the intuition that concisely accommodating a new piece of content into a draft, sometimes require changing the expression of the original content in the draft.

In addition to revision rules, phrase planning and lexicalization rules were also acquired during the corpus analysis. All three were compiled using a corpus analysis methodology presented in Section 2.1. This methodology is applicable to any domain. It allows basing the definition of the range of expressive options that need to be implemented in the generator on systematic empirical data.

1.2.3 An implemented revision-based generator

The third aspect of the research presented in this thesis is the development of the prototype system STREAK implementing the new draft and revision model of language generation. This implementation demonstrates the operationality of both the new generation model and the new type of linguistic knowledge acquired during the corpus analysis.

For this implementation, I built on a pre-existing software environment dedicated to the development of language generation systems: the FUF/SURGE package [Elhadad 1993a] [Elhadad 1993b]. FUF (Functional Unification Formalism) is a programming language based on functional unification [Kay 1979][16]. Both the input and the output of a FUF program are features structures called Functional Descriptions (FDs). The program itself, called a Functional Grammar (FG), is also a feature structure, but one which contains disjunctions and control annotations. The output FD results from the unification of this FG with the input FD. The disjunctions in the FG make unification non-deterministic.

Since it implements the model of Fig. 1.7, STREAK consists of the four components shown in that figure. Each one of them consists of an interpreter and a declarative knowledge source for a total of eight modules. The FUF/SURGE package provided three out of four interpreters and most of one the knowledge sources.

The interpreters for the phrase planner, lexicalizer and syntactic grammar rely on the top-down recursive unification mechanism built-in in FUF. This mechanism is inherently monotonic. Since the reviser needs to perform the non-monotonic revision operations identified during the corpus analysis, I developed a different interpreter for the reviser. It relies on new cut and paste operations implemented using system-level FUF functions[17].

FUF comes as a package with SURGE, a syntactic grammar of English usable as a front-end portable across generation applications. The version of SURGE available when I started the implementation of STREAK had a wide-coverage at the simple clause and determiner ranks but not at the complex sentence and nominal ranks. Drawing both on a variety of non-computational descriptive linguistic works such as [Quirk *et al.* 1985] and on the syntactic data compiled during the corpus analysis, I extended SURGE at these two ranks. At the complex sentence rank, the coverage of the resulting SURGE-2.0 version of the grammar goes way beyond the specific sports sublanguage needed for STREAK.

These two extensions to the FUF/SURGE package (the revision rule interpreter and SURGE-2.0) were essentially preparatory work, paving the way for the core implementation of STREAK. This core consisted of encoding the linguistic data compiled during the corpus analysis as three declarative knowledge sources: the phrase planning rule base, the lexicalization rule base and the revision rule base (each represented as an FG).

Both the example of incremental complex sentence generation given in Section 1.2.1 and the example of syntactic paraphrasing given in Fig. 1.5 are actual runs of STREAK. Note the non-monotonic revisions from steps 1 to 2 and 3 to 4 where the verbs (in italics) "to rout" and "to register" are respectively deleted,

[16] FUF is implemented in Common Lisp.

[17] The most complex of these functions have been implemented specially for the development of STREAK by FUF's creator M.Elhadad, to whom I am therefore deeply indebted.

without loss of content, to make room for additional background facts.

1.2.4 A quantitative evaluation of the opportunistic model and revision rules

The fourth and last aspect of the research presented in this thesis is a quantitative evaluation. It estimates the same-domain robustness and cross-domain portability of both the new generation model and the new linguistic knowledge structures on which this model is based. The initial corpus from which the linguistic knowledge structures were acquired covered a season of basketball reporting. To assess their robustness I used two subsequent years of basketball reporting as test corpora. To assess their portability across domains I used a corpus of stock market reports as a test.

I evaluated two aspects of robustness: coverage and extensibility. I first defined a set of parameters assessing the *coverage* of the target sublanguage for the application domain of STREAK, that could be attained by analyzing a one year sample of that sublanguage. Different parameters were used for measuring the impact, on such coverage limit, of using:

- The knowledge structures needed by the one-pass, macrocoded model of previous systems generating multi-clause sentences.
- The knowledge structures needed by the draft and revision, microcoded model proposed in this thesis.

I also orthogonally defined different parameters for different types of coverage (conceptual, rhetorical and paraphrasing power) in addition to the overall realization coverage. I then defined a set of similar parameters, but this time for measuring the *extensibility* of the respective approaches. The coverage parameters answered the question: with the knowledge structures acquired by analyzing a year sample of the sublanguage, how many sentences from a different year sample could be generated? In contrast, the extensibility parameters answered the question: how many new knowledge structures are needed by a generator that was based on one year sample, in order to also fully cover a different year sample? The results of this evaluation given in Section 5.2.5 shows that the new generation model proposed in this thesis dramatically pushes back the coverage limit of the sublanguage studied. It also, though less spectacularly, improves extensibility.

Having established the same-domain robustness of a generator using the revision rules acquired from the original sports domain, I then evaluated the portability of these rules to another domain. I thus carried out on a corpus of stock market reports the same pairwise analysis that I used to abstract these rules on the original corpus. For example, consider again the sentence pair $< S_1^a, S_0 >$ of Fig. 1.8 reproduced below:
S_1^a: ... the Utah Jazz **handed** the Boston Celtics **their sixth straight home** *loss* ..."
S_0: ... the Chicago Bulls *defeated* the Cleveland Cavaliers ..."

from which the revision rule `Nominalization with Ordinal Adjoin` was abstracted.

The following phrase pair $< S_1^b, S_0^b >$ from the test corpus shows that the same rule can be used for incrementally generating stock market reports as well:
S_1^b: "... the Tokyo stock market **posted its third consecutive** *decline*"
S_0^b: "... the Tokyo stock market *plunged*"

As in the case $< S_1^a, S_0 >$ above, the result, verbally expressed in S_1^b by "*to plunge*" is nominalized as "*its decline*" in order to be subsequently pre-modified by the ordinal "*third consecutive*" which conveys the additional streak information.

The revision rules were classified as a hierarchy. The portability evaluation shows that 69.5% of the branches in this hierarchy are portable. Given the conceptual distance between the sports and financial domains, these results suggest that both the generation model and linguistic data presented in this thesis could be successfully reused to develop summarization systems in multiple quantitative domains such as those listed at the beginning of this introduction.

1.3 Contributions

The research presented in this thesis, makes eight contributions to the field of natural language generation:

- Conveying the historical context in report generation.
- Planning and realizing complex sentences.
- Making the generation process more flexible by allowing decisions to be made opportunistically and under a wider range of constraints.
- Making the generation process more compositional by building sentences incrementally and from individual words.
- Abstracting revision operations for incremental generation.
- Extending SURGE to complex sentences and quantitative nominals.
- Quantitatively comparing generation models
- Quantitatively evaluating the portability of knowledge structures used for generation

I briefly elaborate on each of them in the rest of this section.

Conveying the historical context in report generation STREAK is the first summary generator to provide the historical background of the new facts it reports. It thus not only *summarizes* a particular event but also *contextualizes* it. This constitutes a significant step towards bridging the gap between computer-generated reports and their human-generated counterparts. Conveying historical information allows for a much broader coverage. A generator ignoring such information could at best cover 35% of the sentences from the corpus of human-written summaries I analyzed. Taking into account the historical context also allows for a smarter choice of the new facts to report, since the relevance of a new fact is largely dependent on its historical significance.

Planning and realizing complex sentences STREAK is the first generator that deliberately attempts to pack as many facts as possible within a single sentence. As a result, STREAK generates sentences up to the maximum complexity level observed in human-written summaries (46 words long, 10 level deep, conveying 12 facts). These sentences are significantly more complex than those generated by any previous system. The development of STREAK required addressing for the first time at the sentence rank, the full blown content planning issues previously investigated only at the paragraph rank.

Making the generation process opportunistic and more flexible STREAK is the first system to handle obligatory content prescriptively and supplementary content opportunistically and under surface form constraints. This makes the generation process far more flexible. In particular, it allows using linguistic factors to monitor content planning, a requirement when concisely expressing the content of a whole paragraph in a single sentence without letting that sentence grow so complex as to be unreadable.

Making the generation process more compositional and scalable STREAK generates sentences more compositionally than any previous generators. It is compositional in three orthogonal ways: by building sentences from individual words, by separating the organization of content units inside the sentence from the lexicalization of the individual units and by building sentences incrementally starting from a basic draft into which additional content units are gradually incorporated. This extreme compositionality makes STREAK more easily scalable and portable.

Abstracting revision operations for incremental generation The hierarchy of revision operations presented in this thesis constitutes a new type of linguistic knowledge. It is the first set of linguistic resources to be simultaneously based on an extensive corpus analysis and specifically geared towards incremental language processing. Their operationality is demonstrated by their implementation in the revision rule base of STREAK. Their relevance to other quantitative domains is shown by their high degree of portability from the domain where they were acquired to a new domain.

Extending SURGE to complex sentences and quantitative nominals SURGE is easy to use thanks to the uniform, bi-directional and declarative formalism of FUGs on which it is based. The extensions I made to SURGE at the nominal and complex sentence ranks, endow it with wide coverage at all four sentential linguistic ranks: determiner, nominal, simple clause and complex sentence. This extended coverage combined with its easy of use, makes SURGE the best portable syntactic processing front-end for the development of generation applications available today.

Quantitatively comparing generation models The robustness evaluation I carried out is the first attempt to quantify how much of a given sublanguage can be captured by various knowledge structures used for language generation and acquired from samples of this sublanguage. It is also the first quantitative comparison of different generation models. It establishes the superiority of the new microcoded revision-based generation model over the traditional one-pass macrocoded generation model of previous systems for the generation of multi-clause sentences. There is hardly any previous work in quantitative evaluation methods for generation.

Quantitatively evaluating the portability of knowledge structures used for generation The portability evaluation of the revision operations described in this thesis is the first attempt to quantitatively assess the domain-independence of knowledge structures used for generation and acquired from texts in a single domain. It establishes the relevance of both these revision operations and the opportunistic generation model that relies on them to other quantitative domains.

1.4 Guide to the rest of the thesis

In the next chapter, I describe the original corpus analysis that resulted in the hierarchy of revision operations needed by the new generation model proposed in this thesis. In the subsequent chapter, I present in detail a complete system architecture for text generation based on this model. In Chapter 4, I turn to the implementation of this model in the system STREAK. In Chapter 5, I present the quantitative evaluations of the same-domain robustness and cross-domain portability of this revision-based generation model. In Chapter 6, I compare the research presented in the previous chapters to related work in the fields of summary report generation, revision-based generation, incremental generation and evaluation in generation. In Chapter 7, I conclude by revisiting the contribution this thesis makes to each of these fields, and discussing its limitations and future directions within the new research framework that it defines. In Appendix A, I present in full detail the revision rule hierarchy abstracted from the corpus analysis of Chapter 2. In Appendix B I describe the FUF/SURGE package underlying the implementation of STREAK, including the extensions to the syntactic grammar. In Appendix C I provide a set of example runs of STREAK, some of them together with the full input and intermediate representations used by the system during the run. Finally, Appendix D gives an overview of CREP a software tool that I used[18]. for partially automating the various corpora analysis that I performed during the knowledge acquisition and evaluations stages of this research.

[18] And designed in collaboration with Darrin Duford who implemented it

Chapter 2

Corpus analysis of newswire reports

2.1 Motivation and methodology

In this chapter I present a corpus analysis of newswire reports summarizing basketball games[1]. An example report from this corpus was given in Fig. 1.1 of Section 1. Another example is given in Fig. 2.1 in Section 2.2. The corpus consisted of 833 reports covering an entire professional basketball season. Almost all of them contained some historical information.

The overall goal of this corpus analysis was threefold:

- Identify the set of linguistic resources available to convey specifically historical information.
- Provide criteria for a generation architecture able to concisely and flexibly convey historical information.
- Provide data for the knowledge sources of a prototype system based on such an architecture.

The core of this analysis involved distinguishing the different types of information expressed in the corpus, and for each of these types, the linguistic forms used for its expression. Such a detailed *semantic* analysis can only be carried out empirically by hand[2]. It was decomposed in five successive steps, each with its particular goal:

1. *Focusing on a restricted category of corpus sentences.* This restricted category defined both a sub-corpus for systematic and in-depth analysis and a realistic target output for the prototype system STREAK.
2. *Defining the ontology of the sub-corpus.* This involved identifying the different types of entities mentioned in the corpus[3] as well as the different types of facts expressed about these entities.
3. *Identifying the semantic constraints on concept grouping inside the corpus sentence constituents.* These constraints can be viewed as schemata for phrase level content organization. In STREAK, they are implemented in part in the phrase planning rules and in part in the revision rules.
4. *Identifying the various syntactic forms used in the corpus to express each concept combination.* This resulted in a set of mappings from semantic to syntactic structures. These mappings, which I call *realization patterns*, capture the paraphrasing power of the domain sublanguage. In STREAK, they are implemented in part in the lexicalization rules and in part in the revision rules.

[1] These reports were taken from the UPI newswire.

[2] I use corpus analysis as a methodology for semantic and syntactic knowledge acquisition. My work thus pertains to the *knowledge-based* tradition of NLP research and *not* to the statistical tradition of NLP research that is at times glossed as "corpus-based NLP" in its most recent revival. Statistical NLP attempts to build systems without any knowledge but which instead rely on correlation between their input and occurrence probabilities computed on huge textual corpora.

[3] Where there is no risk of ambiguity, I will refer to the *sub*corpus simply as "the corpus".

5. *Identifying a set of revision tools to attach floating facts to basic sentence structures as well as the semantic, syntactic and lexical constraints on their applicability.* This was carried out by a pairwise comparison of realization patterns differing by only one concept. Each revision tool corresponds to a class of structural transformations to map the simpler realization patterns into the more complex patterns with one more concept. These revision tools are implemented in the revision rules of STREAK.

These five steps are presented in order in the following subsections. Together, they define a corpus-based knowledge acquisition methodology that allows identifying the range of expressive options that need to be implemented in a generator based on systematic empirical data. This systematism in turn allows thorough testing of the generator and quantitative evaluations of its output. The differential analysis of realization patterns defines an approach to extract from text, linguistic knowledge, that is specifically geared towards incremental processing, such as revision tools.

2.2 Focusing on a sub-corpus

In order to fulfill its goals, the corpus analysis needed to be simultaneously semantic, syntactic, very fine-grained[4] and systematic. Such an ambitious analysis could not have been carried out on the total corpus, *i.e.,* on each and every sentence of 833 reports. It was necessary to focus on a restricted category of corpus sentence. I defined this restricted category in two steps. I first made observations about the structure of the corpus reports to choose a *discursive* focus. I then distinguished between different types of information conveyed in the corpus to choose a *semantic* focus.

2.2.1 Report structure and discursive focus

Journalism textbooks define a variety of standard report structures. The corpus reports are organized following the so-called *inverted pyramid structure with summary lead* [Fensch 1988], meaning that crucial information is packed in the lead followed by facts of decreasing importance towards the end of the report. Summary type leads, often consisting of a single sentence in the corpus reports, are thus self-contained mini-reports containing the basic facts. This type of report structure is not particular to sports reporting but is pervasive in newswire articles [Mencher 1984]. It is preferred because newswire reports are essentially used as draft material to be edited by client newspapers under heavy time-pressure and stringent space constraints. An inverted pyramid structure with a summary lead makes a report instantly editable by cutting its tail. When space is really scarce only the lead remains.

This particular structure of the corpus reports allowed me to narrow the discursive scope of the analysis to a single sentence per report: the lead sentence. This sub-corpus of lead sentences was of manageable size for a systematic manual analysis and at the same time allowed focusing on the two main linguistic phenomenon of interest in this thesis: summarization and historical contextualization. In the literature about summarization (*e.g.,* [Borko 1975]), an important distinction is made between *indicative* abstracts which provide meta-level information on the text itself (its goal, structure, style etc), and *informative* abstracts which express the *content* of the text in a less detailed and more compact form. Summary leads are informative abstracts of the rest of the report Moreover, 52% of them contained historical information. That such historical background was part of the essential facts in a majority of reports confirms the importance of historical information. It also allowed the detailed investigation on how its gets folded with new information into complex sentences.

2.2.2 Content types and semantic focus

In order to define a semantic focus for the fine-grained analysis, I first had to carry out a high-level semantic analysis of *all* 833 lead sentences in the corpus. This high-level analysis consisted of distinguishing different types of information conveyed in these sentences. Among these different types I singled out one particular

[4]i.e., down to the individual word whenever necessary

type of new information, *box-score facts* and two particular types of historical information, *records* and *streaks*.

A box-score is a table containing a standard set of facts for one game. In daily newspapers, there is one box-score accompanying each game report. It essentially contains data quantifying the various aspects (scoring, rebounding, passing etc) of the total performance of each player and team for the entire game. An example box-score with the corresponding report is given in Fig. 2.1. In this report, fine-grained statistics *not* available in the box-score, such as performances over a short time interval during the game, are emphasized by an italic font (historical information is, as usual, emphasized by a boldface font). Such finer grained statistics come from complete game charts [Anderson 1985]. Box-scores are available on-line through newswire and sport statistic computer services, whereas game charts are not[5]. Therefore, only box-scores constitute a realistic tabular input for a report generation system. This is the first reason to choose box-score facts as the semantic focus for new information. There are three other reasons:

- They are the most common type of new information. It was the sole type of new information in 50.2% of the corpus sentences.[6].
- They are the type of new information most often associated with historical information. 65% of the corpus sentences in which box-score statistics was the sole type of new information also contained historical information. Only 39% of the first sentences containing new information *other* than box-score facts also contained historical information.
- They are not conceptually idiosyncratic to basketball or sports. Other quantitative domains contain statistics that are direct conceptual equivalent of box-score facts in the basketball domain. For instance, the final value of a financial index in the stock market domain (*e.g., "IRT property ended at 11,104"*) directly corresponds an the end of the game statistic in the sports domain (*e.g., "Dikembe Mutombo finished with 19 points"*).

A record is the property of a statistic to constitute a maximum or a minimum in a set of similar statistics over some period of time, e.g., *"Michael Adams scored* **a career-high** *44 points"*. A streak is the property of the game result to extend or interrupt a series of similar results for a given team, e.g., *"the Denver Nuggets* **ended a five-game losing streak with** *a 128-112 victory over the Sacramento Kings"*

There are three reasons for choosing records and streaks as the semantic focus for historical information:

- They can be easily maintained from box-scores (they are also available on-line through sports statistic computer services).
- They are the most common type of historical information (together, records and streaks were the sole type of historical information in 62% of the first sentences).
- They are pervasive in any quantitative domain, as illustrated by the following examples of streaks from finance (*e.g., "the Dow Jones Average of 30 Industrials recorded its fourth straight loss"* and meteorology *"in New York the temperature remained below zero for the fifth consecutive day"*).

Focusing on box-score facts, records and streaks defines a sub-corpus of 293 lead sentences for fine-grained semantic and syntactic analysis. The special properties of these three types of information listed above insure that this sub-corpus:

- Constitutes a representative core of the whole corpus.
- Is ideal to study how historical background gets opportunistically woven into sentences conveying related new facts.
- Covers the most realistically available on-line input data for the domain at hand.
- Should yield results and linguistic knowledge that generalize to other quantitative domains.

[5] To the best of my knowledge.
[6] Note also in Fig. 2.1 that most of the information conveyed in the report comes directly from the box-score.

ORLANDO MAGIC					24	24 (**48**)		21 (69)		25 (**94**)	
		fg	**3pt-fg**	**ft**	**rb**						
Players	**mn**	**m-a**	**m-a**	**m-a**	**o-t**	**bl**	**st**	**as**	**to**	**pf**	**tp**
Catledge	44	8-15	0-0	0-0	4-11	0	3	3	2	3	16
Reynolds	27	3-9	1-1	2-2	0-2	1	1	1	1	3	9
Roberts	16	4-7	0-0	1-2	2-6	2	0	1	1	5	9
Vincent	22	4-12	0-0	0-0	3-6	0	1	3	4	0	8
Anderson	30	6-10	0-0	2-4	1-2	0	1	1	0	2	14
Kite	22	2-4	0-0	0-0	3-4	0	0	0	0	2	4
O.Smith	26	4-11	0-0	1-2	0-1	0	2	1	0	2	9
Skiles	26	8-15	0-3	5-5	1-2	0	**4**	**7**	1	3	**21**
Scott	14	1-7	0-1	2-2	0-0	0	0	0	1	1	4
Je. Turner	4	0-1	0-0	0-0	0-1	0	0	1	0	3	0
Williams	9	0-1	0-0	0-0	0-1	0	0	0	0	0	0
Team totals	240	40-92	1-5	13-17	14-**36**	2	12	18	12	24	**94**
Team %		**.435**	.200	.765							
HOUSTON ROCKETS					20	25 (**45**)		31 (76)		23 (**99**)	
		fg	**3pt-fg**	**ft**	**rb**						
Players	**mn**	**m-a**	**m-a**	**m-a**	**o-t**	**bl**	**st**	**as**	**to**	**pf**	**tp**
Johnson	38	4-10	0-1	2-4	2-8	1	2	4	1	3	10
Thorpe	40	5-11	0-0	4-4	5-**10**	0	0	2	1	3	**14**
Olajuwon	33	6-16	0-0	5-6	3-**14**	**8**	2	4	3	**6**	**17**
Maxwell	42	**4-14**	3-9	3-4	0-3	0	1	5	3	3	**14**
K.Smith	38	**12-15**	1-1	3-3	1-4	0	2	**9**	6	0	**28**
Rollins	14	0-0	0-0	0-0	1-2	1	0	0	0	2	0
Floyd	19	2-5	1-2	7-8	0-3	0	0	4	4	2	12
Herrera	8	1-3	0-0	2-4	0-1	0	0	0	1	2	4
Jo. Turner	5	0-0	0-0	0-0	0-1	0	0	0	1	2	0
Bullard	3	0-0	0-0	0-0	0-0	0	0	0	0	0	0
Team totals	240	34-74	5-13	26-33	12-**46**	10	7	28	20	21	**99**
Team %		.549	.385	.788							

ORLANDO, Fla. (UPI) – Kenny Smith scored 28 points Sunday night to pace the Houston Rockets to a 99-94 victory over Orlando, **giving the Magic their league-high 10th straight loss**.
Hakeem Olajuwon contributed 17 points, 14 rebounds and eight blocked shots before fouling out *with 3:15 remaining in the game*.
The Magic led 48-45 at halftime, *but Houston outscored Orlando 23-12 in the first eight minutes of the third quarter to take the lead for good*.
Smith converted 12 of 15 shots from the field and dished out nine assists **to give the Rockets their sixth win in eight meetings with Orlando**.
Scott Skiles provided spark of the bench with 21 points, seven assists and four steals for the Magic, **which lost for the 14th time in 15 games**.
Otis Thorpe added 14 points and 10 rebounds for the Rockets. Vernon Maxwell scored 14 points despite 4-for-14 shooting from the field.
Orlando was out-rebounded 46-36 and shot just 43.5-percent from the field. **The Magic have dropped their last six home games**.

Font conventions in the report: *italics = information not coming from the box-score*, **boldface = historical information**.
Font conventions in the box-score: **boldface = statistic included in the report**.
Abbreviations in the box-score : mn = MiNutes-played, fg = Field-Goals, ft = Free-Throws, rb = ReBounds, as = ASsists, st = STeals, bs = Blocked Shots, to = TurnOver, pf = Personal Foul, tp = Total Points, 3pt = three-PoinT, m = Made, a = Attempted, o = Offensive, t = Total..

Figure 2.1: Standard box-score and corresponding natural language report

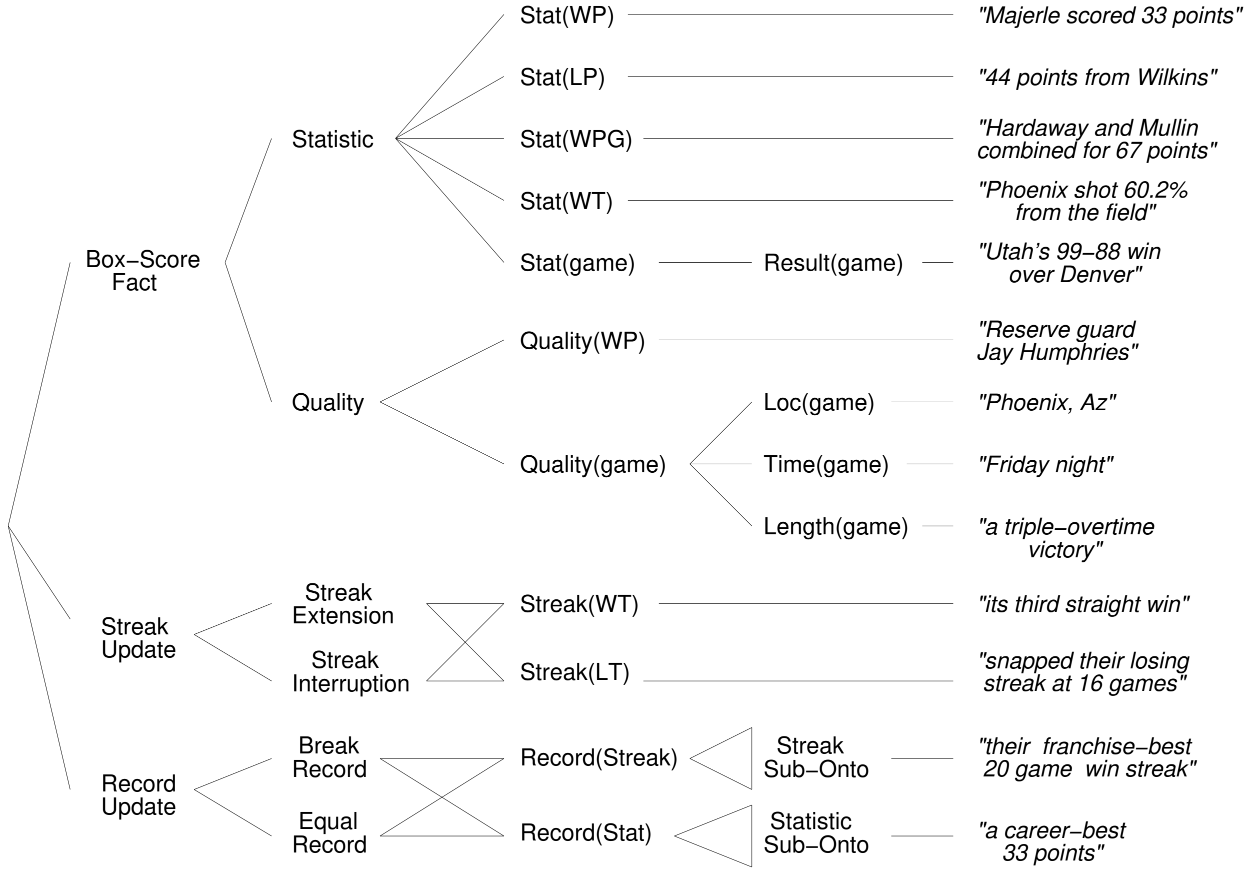

Figure 2.2: Domain ontology

This sub-corpus also defines the sub-domain for the prototype system STREAK. The ontology of this sub-domain is presented in the next section.

2.3 Domain ontology

In this section I distinguish and exemplify the various types of information conveyed in the sub-corpus defined in the previous section. The top-levels of the ontology defined by these subtypes is shown in Fig. 2.2.

At the top level categories in this ontology are the three major classes of facts that defined the semantic scope of this sub-corpus: one class of new facts, box-score facts, and two classes of historical facts, records and streaks about these box-score facts.

Box-score facts first specialize into statistics and qualities. Then, each category further specializes in terms of the domain entity they concern: either an individual player, a group of player, an entire team, or the game as a whole. Statistics about individual players specialize into those concerning players of the winning team, `stat(WP)`, and those concerning players of the losing team, `stat(LP)`. Each case is further refined in terms of unit (points, rebounds, assists etc.). The only cases of statistics concerning either a group of players, `stat(WPG)`, or a whole team, `stat(WT)`, were restricted to the winning team. Corresponding data about the losing team was never deemed important enough in the appear in the lead sentence. Similarly, only qualities concerning players from the winning team, `quality(WP)`, were conveyed in the lead. The only statistic about the whole game is its result, specifying the winner, loser and final score. Though per-quarter scores are available in standard box-scores, none of those appeared in the lead sentences. Qualities about the game include its location, time, and length.

Streak updates specialize along two orthogonal dimensions, streak extensions *vs.* streak interruptions, and streak of the winning team, `streak(WT)`, *vs.* streak of the losing team, `streak(WT)`. Similarly, record updates specialize into breaking *vs.* equaling types and orthogonally into records of game statistics and records of streak lengths. Record game statistics are further refined in terms of the various types of game statistics from the sub-ontology of game statistics defined above. Similarly, record streaks are further refined in terms of the sub-ontology of streaks.

The bottom level of the ontological hierarchy where all low-level distinctions differentiating various statistics are taken into account, contains 42 concepts.

2.4 Corpus sentence structures

2.4.1 Top-level structure: concept clustering

The semantic structure of corpus sentences is defined by constraints on concept co-occurrence, *i.e.,* what concept combinations appear inside particular sentence constituents. Their surface structure is defined by constraints on syntactic dependency between the constituents realizing the various elements of each appearing concept combination.

I made two main observations on concept co-occurrence inside whole corpus sentences. The first observation is that the number of facts in these sentences varies from a minimum of four to a maximum of 12. The second observation is that each sentence contains one instance of each of the four following concepts:

- Game result, `result(game)`
- Game location, `loc(game)`
- Game time, `time(game)`
- Winning player statistic, `stat(WP)`

A game has only one result, location and time. In contrast, most corpus sentences contain several instances of the concept winning player statistic. However, only one of these winning player statistics is the focus of the sentence. I call this particular statistic the *main statistic*, `stat1(WP1)`, of the concept combination.

The conclusion of these two observations is that any concept combination appearing in a corpus sentence is made of:

- The *basic* concept combination: <`stat1(WP1), result(game), loc(game), time(game)`>.
- From zero to eight *additional* concepts from the domain ontology presented in Section 2.3

At the abstract concept combination level, all corpus sentences are thus semantic supersets of the basic, four concept sentences.

To discover the constraints on syntactic constituent dependencies inside the corpus sentences, I first looked at these basic sentences. I observed that they all follow one of only two high-level syntactic structures. These structures are illustrated on two example sentences in Fig. 2.3. In this figure sentence constituents are represented by boxes with their concept at the top and their text at the bottom. These boxes are linked by lines showing the dependency relations between the constituents. In both these sentences the structural status of the syntactic constituents that respectively express the location and the time of the game is identical. The location is expressed by a constituent prefixed to the rest of the sentence as a parenthetical, while the time is expressed by a floating adjunct that is attached to different constituents in different corpus sentences. What sets these two structures apart is the relation between the constituents that respectively express the main statistic and the game result. In the first structure they are in *parataxis* while in the second they are in *hypotaxis*, with the main statistic as head. In this paper, I use the notions of parataxis and hypotaxis defined in [Halliday 1985], because they are general relations between syntactic constituents occurring at *all* linguistic ranks (sentence, clause, group). Two constituents are in parataxis if they are both at the same structural level. Parataxis is thus a general symmetric relation covering both coordination and apposition. In contrast, two constituents are in hypotaxis if it is possible to distinguish one as a main element and the other as a dependent element. Hypotaxis is thus a general asymmetric relation covering both head-argument dependency and head-adjunct dependency. I take clause subordination to be a special case of hypotaxis where both the head and the dependent are clauses.

When I looked at the complex corpus sentences, I observed that they have the same structural characteristics as the basic sentences. In these complex sentences, the location is still expressed as a parenthetical prefix, the time is still expressed by a floating adjunct, there are still two top-level constituents - one including the main statistic and one including the game result - and these two constituents are still either in parataxis or in hypotaxis with the constituent including the main-statistic as head. The only difference from the basic sentence structures is that additional concepts are now grouped with the main statistic and/or with the game result in these two top-level constituents.

An example of a hypotactically structured complex sentence is shown in Fig. 2.4. This sentence contains nine facts: the four basic facts (with their concept highlighted in bold), three historical additional facts (in dashed boxes) and two non-historical additional facts. Compare this sentence with the basic sentence at the bottom of Fig. 2.3. They both share the same high-level structure: hypotaxis with the constituent including the main statistic as head and the constituent including the game result as dependent. In the complex sentence, however, the head constituent contains four additional facts and the dependent constituent contains one additional fact. The corpus also contained paratactically structured complex sentences with additional facts grouped with the main statistic and/or with the game result.

In short, the top-level structure of any corpus sentence is one of the two basic structures illustrated in Fig. 2.3, with additional facts *clustered* either around the main statistic or around the game result. Any corpus sentence is thus made of two halves. In the rest of this thesis, I call the half containing the main statistic the *statistic cluster* and the half containing the game result the *result cluster*. A systematic analysis of these two clusters revealed that whether a fact is part of the statistic cluster or the result cluster is directly dependent on its ontological class. Among the classes of additional facts from the domain ontology shown in Fig. 2.2 only one - `stat(WPG)` *i.e.,* statistic of winning player group - was part of the statistic cluster in some corpus sentences and part of the result cluster in some others. The instances of all the other classes consistently appeared in the same cluster. This concept clustering phenomenon is illustrated in Fig. 2.5. It shows where additional instances of each concept gets attached. These semantic constraints on concept clustering can be viewed as schemata [McKeown 1985] for sentence-rank planning.

Figure 2.5 also shows that the overall structure of any corpus sentence depends on only three factors:

- The top-level relation between the two clusters, either parataxis or hypotaxis with the statistic cluster as head.
- The internal structure of its statistic cluster
- The internal structure of its result cluster

These corpus observations show that any complex sentence can indeed be generated in two steps: (1) produce a basic sentence realizing the obligatory content units, (2) incrementally revise it to incorporate supplementary content units. Furthermore, they indicate that supplementary content units can be attached within a cluster, based on *local* constraints, thus simplifying both generation and the rest of the corpus analysis. When I pursued the analysis inside each cluster, I split the whole sentence corpus into two subsets: one containing statistic clusters and the other, result clusters. This cluster internal analysis is discussed in the next section.

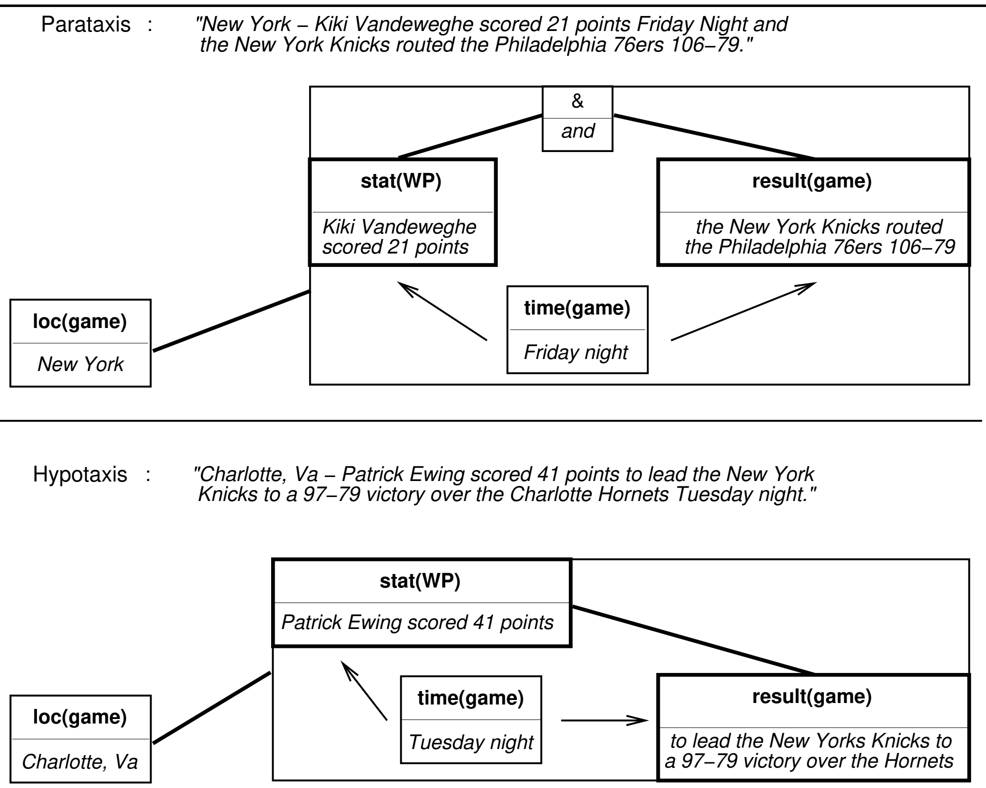

Figure 2.3: Top-level structures of basic sentences

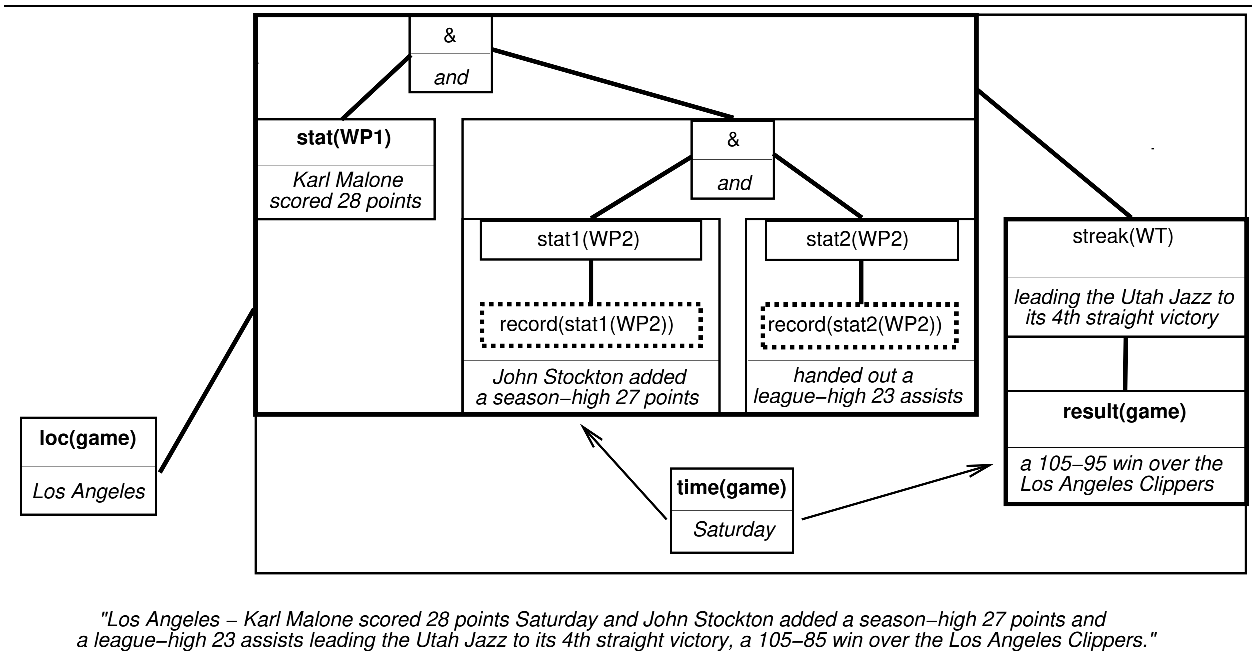

Figure 2.4: Complex sentence with top-level hypotactic structure

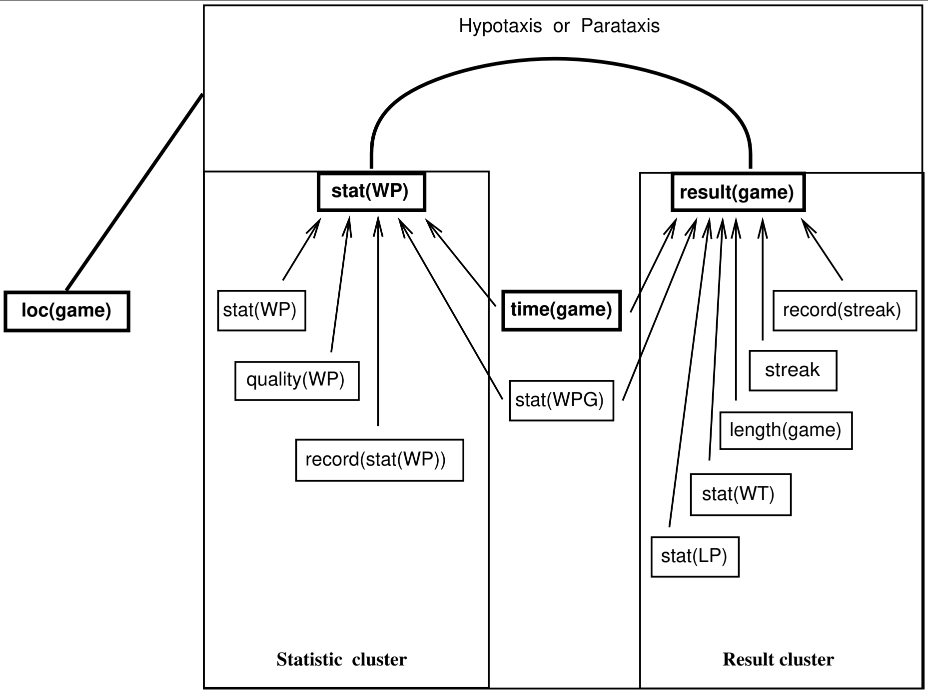

Figure 2.5: Concept clusters

2.4.2 Internal cluster structure: realization patterns

The analysis of the top-level corpus sentence structure presented in the previous section, resulted in a set of *semantic* constraints to guide sentence *planning*. These semantic constraints define the set of concept combinations observed in specific corpus sentence constituents that I called clusters. In order to identify *linguistic* constraints on sentence *realization*, I then looked, for each clustered concept combination, at the set of syntactic structures realizing it in the corpus. I call a mapping from a concept combination onto a syntactic structure a *realization pattern*.

Four examples of realization patterns are given in Fig. 2.6. I represent realization patterns as arrays, with each column corresponding to a syntactic constituent and each row corresponding to some level of information about this constituent[7]. The first row indicates the semantic element that the constituent realizes; the second row its grammatical function; the third row its structural status (i.e. head, argument, adjunct or conjunct) and the fourth its syntactic category[8]. Below each realization pattern, I give two example phrases from the corpus illustrating the pattern.

Realization patterns capture the *structural* paraphrasing power of the corpus sublanguage[9]. In Fig. 2.6, patterns R_{d1} and R_{d2} capture two alternatives ways to convey a game result. Patterns R_{i1} and R_{i2} capture two alternative ways to convey a game result together with a streak update that this result triggers. These patterns abstract from lexical material and syntactic details (*e.g.,* connectives, mood) to focus on representing the different mappings from semantic structure to syntactic structure. Consider for example the two alternative patterns R_{i1} and R_{i2} given in Fig. 2.6 for the realization of the same concept combination C_i.

[7] Some rows are double. A single line separates the two halves of a double row, whereas a double line separates two different rows.

[8] The particular grammatical functions, structural relations and syntactic categories used in this thesis are those of SURGE and are described in detail in Appendix B

[9] But abstracts from the more productive grammatical and lexical paraphrasing, cf. Section 3.3.1.5 for a detailed discussion of these different types of paraphrasing.

C_i: <game-result(winner,loser,score),**streak(winner,aspect,type,length)**>.
C_d: <game-result(winner,loser,score)>.

$R_{i1}(C_i)$:

winner	game-result	loser	score			**length**	**streak+aspect**	**type**
agent	process	affected	score	frequency				
arg	head	arg	adjunct	adjunct				
proper	verb	proper	number	PP				
				prep	[det	ordinal	adj	noun]
Chicago	beat	Phoenix	99-91	**for**	**its**	**3rd**	**straight**	**win**
New York	defeated	Seattle	101-91	**for**	**its**	**4th**	**consecutive**	**victory**

$R_{d1}(C_d)$ surface decrement of $R_{i1}(C_i)$:

winner	game-result	loser	score
agent	process	affected	score
arg	head	arg	adjunct
proper	verb	proper	number
Seattle	defeated	Sacramento	121-93
Detroit	routed	Boston	110-89

$R_{i2}(C_i)$:

winner	**aspect**		**type**	**streak**	**length**			score	game-result	loser
agent	process	affected/located			location	instrument				
arg	head	arg			adjunct	adjunct				
proper	verb	NP			PP	PP				
		det	participle	noun		prep	[det	number	noun	PP]
Utah	**extended**	**its**	**winning**	**streak**	**to 6 games**	**with**	a	118-94	triumph	over Denver
Boston	**stretching**	**its**	**winning**	**spree**	**to 9 outings**	**with**	a	118-94	rout	of Utah

$R_{d2}(C_d)$ surface decrement of $R_{i2}(C_i)$:

winner			score	game-result	loser
agent	process	range			
arg	head	arg			
proper	support-verb	NP			
		det	number	noun	PP
Chicago	claimed	a	128-94	victory	over New Jersey
Orlando	recorded	a	98-87	win	against Dallas

Figure 2.6: Realization pattern examples

In these two patterns, the semantic elements `streak` and `aspect`[10] are mapped onto syntactic constituents in drastically different ways. In pattern R_{i1}, they are both expressed by an adjective (*e.g., "straight"* or *consecutive*) inside a purpose adjunct PP. In contrast, in pattern R_{i2}, `aspect` is expressed by a head verb (*e.g., "to extend"* or *"to stretch"*) and `streak` by a head noun (*e.g., "streak"* or *"spree"*) of the object NP.

I identified the realization patterns of each clustered concept combinations, finding a total of 160 patterns for 79 combinations. Each of these realization patterns can be obtained by applying a set of information-adding revisions to one of the three realization patterns of the basic four concept combination. In this thesis, instead of going through the list of the patterns resulting from these revisions, I focus on the more concise presentation of the revision operations themselves. Example of these operations are given in the next section. Their exhaustive list is given in Appendix A.

2.5 Revision tools

2.5.1 Differential analysis of realization patterns: identifying revision tools

Realization patterns specify syntactic structures for *entire* concept clusters. For example, consider again the complex sentence from the corpus whose structure was analyzed in Fig. 2.4:
"Karl Malone scored 28 points and John Stockton added a season-high 27 points and handed out a league-high 23 assists leading the Utah Jazz to its fourth straight victory, a 105-95 win over the Los Angeles Clippers".

This sentence contains two realization patterns:

- One pattern for its five fact statistic cluster *"Karl Malone scored 28 points and John Stockton added a season-high 27 points and handed out a league-high 23 assists"*.
- Another pattern for its two fact result cluster *"leading the Utah Jazz to its fourth straight victory, a 105-95 win over the Los Angeles Clippers"*.

Such realization patterns are thus too coarse grained to be usable by a revision-based generator exploiting compositionality inside clusters. The linguistic data needed by such a system is a set of *sub*patterns specifying the realization of a *single* additional concept in the context of a pre-existing draft cluster structure. For example, one such subpattern would specify that in the context of a draft game result cluster like *"leading the Utah Jazz to a 105-95 win over the Los Angeles Clippers"* an additional streak concept can be realized by an apposition like *"its fourth straight victory"*.

In order to identify these contextual sub-patterns I carried out a differential analysis of the cluster realization patterns. A pictorial description of this differential analysis is given in Fig. 2.7. This analysis is based on the notions of *semantic decrement* and *surface decrement*. A cluster C_d is a semantic decrement of cluster C_i if C_d contains all but one of C_i's concepts. Each cluster has a set of realization patterns associated with it. The surface decrement of a realization pattern of C_i is the realization pattern of C_d that is structurally closest. Structural distance is defined as the number of matching cells in the arrays representing their respective realization patterns. In cases of tie, the cells of some rows, such as the one for the constituent realizing the semantic head, are given more weight.

Figure 2.6 shows a semantic decrement pairing C_d, a single concept, with C_i, which contains two concepts. Both clusters have two realization patterns associated with them as they each can be realized by two different syntactic structures. These four syntactic structure patterns must be compared to find the surface decrements. Since R_{d1} is entirely included in R_{i1} and R_{d2} is not, R_{d1} is the surface decrement of R_{i1}. Such a case of inclusion is the simplest type of relation between a pattern and its surface decrement. However, the maximum overlap between a realization pattern and its decrement is sometimes only partial. To identify the

[10] The "aspect" of a streak update differentiates streak extensions from streak interruptions. It is an entirely different notion from the grammatical aspect of a clause.

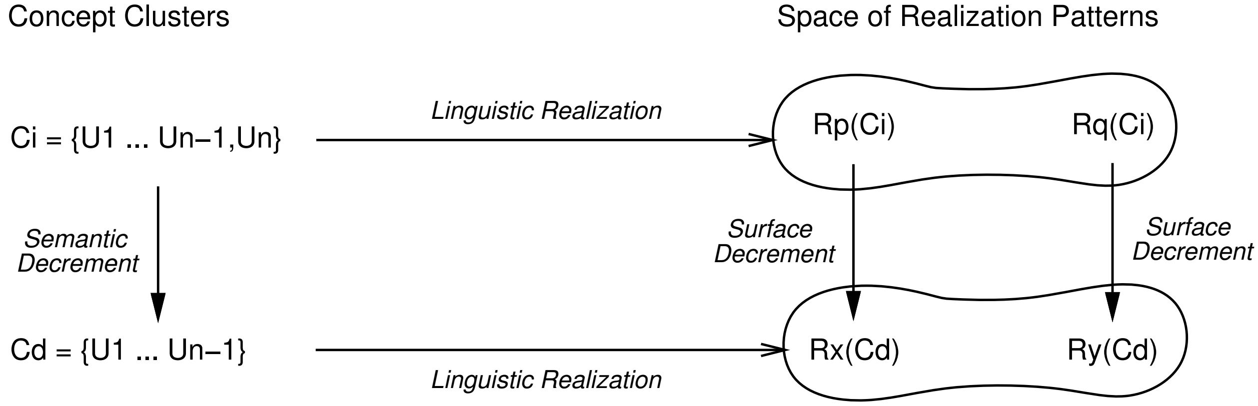

Figure 2.7: Differential analysis of realization patterns

surface decrement of R_{i2} for example, we need to compare it to R_{d2} and R_{d1} in turn. In R_{d1}, only the first column matches a column in R_{d2}. In contrast, in R_{d2}, in addition to the first column, columns 3 to 6 also (if only partially) match some column in R_{d2}. In particular, the semantic head, `game-result`, is mapped onto a noun in both R_{d2} (*e.g., "victory"*) and R_{i2} (*e.g., "triumph"*), whereas in R_{d1} it is mapped onto a verb (*e.g., "to defeat"*). Therefore, it is R_{d2} rather than R_{d1} that is the surface decrement of R_{i2}. However, R_{d2} only partially overlaps R_{i2} since its second column does not correspond to any column in R_{i2}.

By systematically listing the surface decrements (and conversely the surface increments) of each of the 160 realization patterns resulting from the previous step of the analysis, I identified 270 surface decrement pairs. I then considered each of these 270 pairs as an instance of revision. The simplest pattern in the pair is the *base* pattern of the revision and the most complex pattern in the pair the *revised* pattern. For example in Fig. 2.6, patterns R_{d1} and R_{d2} play the role of base patterns, whereas patterns R_{i1} and R_{i2} play the role of revised patterns.

For each revision instance, there is a set of structural transformations to change the base pattern into the revised pattern. Each such set of structural transformations defines a revision tool[11]. In the last step of this corpus analysis I classified revision tools in terms of their structural characteristics. This last step is presented briefly in Section 2.5.2 and in full detail in Appendix A.

2.5.2 Classifying revision operations

Classifying the structural differences between surface decrement pairs resulted in a hierarchy of revision operations. The top-level of this hierarchy, shown in Fig. 2.8, distinguishes between *monotonic* revisions, which are abstracted from fully overlapping decrement pairs and involve only attachment of a new constituent, and *non-monotonic* revisions, which are abstracted from partially overlapping decrement pairs and also involve displacement and/or deletion of draft constituent(s). Monotonic revisions consist of a single transformation which preserves the base pattern and adds in a new constituent. In non-monotonic revisions an *introductory* transformation breaks up the integrity of the base pattern in adding in new content. Subsequent *restructuring* transformations are then necessary to restore grammaticality. Monotonic revisions can be viewed as elaborations while non-monotonic revisions require true revision. In the next two sections, I discuss and give examples of these two different classes of revision tools.

[11] The transformations involved in the application of a revision tool are not related to the notion of transformation as defined by transformational grammarians. In transformational grammars, a transformation relates surface forms that are grammatical paraphrases of *a common* deep semantic form. In contrast, a transformation involved in the application of a revision tool maps a surface form S_1 realizing a deep semantic form D_1 onto a surface form S_2 realizing *a distinct* deep semantic form D_2, which contains D_1 *plus an additional fact.*

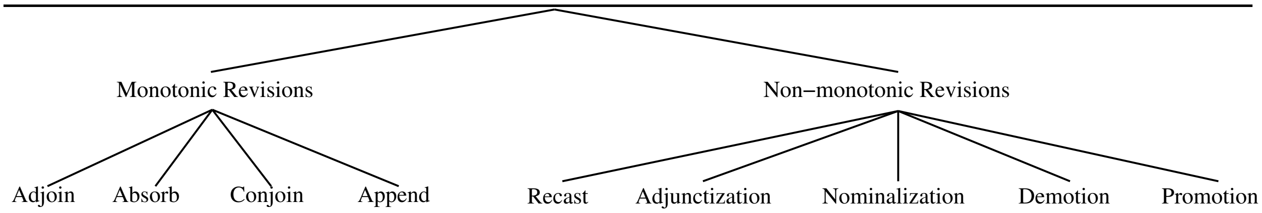

Figure 2.8: Revision operation hierarchy: top-levels

2.5.2.1 Monotonic revision

I identified four main classes of monotonic revisions: `Adjoin, Append, Conjoin` and `Absorb`. They differ from each other in terms of either the type of the base structure on which they can be applied or the type of revised structure they produce.

For example, `Adjoin` applies only to hypotactic base patterns. It consists of the introduction of an additional optional constituent A_c. A_c is thus *adjoined* to the base constituent under the base head B_h. The revision schema of an adjoin is shown in Fig. 2.9[12].

Like most monotonic revision tools, `Adjoin` is versatile. The variety of `adjoin` revision operations is shown in Fig. 2.10. The analyzed corpora contained cases where the new constituent was added to a nominal (abbreviated NP in the revision hierarchy) and others where it was added to a clause (abbreviated S in the revision hierarchy). When adjoined to a nominal, the new constituent could fill the following syntactic functions: partitive, classifier, describer and qualifier. For the qualifier syntactic function the added constituent came in two syntactic forms: non-finite clause and relative clause. Finally, a relative clause could express a given type of additional information equally well when adjoined to different draft subconstituents of the same syntactic category (nominal) but embedded at different levels in the draft structure. Consider for example, adding streak information to the draft phrase:
"to power the Golden State Warriors to a 135 119 triumph over the Los Angeles Clippers",

The same revision tool `Adjoin Relative Clause to Nominal` can be applied to the embedded nominal referring to the losing team (underlined) and yielding:
"to power the Golden State Warriors to a 135 119 triumph over [[the Los Angeles Clippers] , [**who lost for the ninth straight time.**]]"

[12]The pictorial conventions used in this figure, and in Fig. 2.11 are the following: constituents are circles or ovals, structural relationships are lines, the lines corresponding to role relations (either argument or adjunct) are labeled. The elements **added** by the revision, either constituents or relations, are boldfaced.

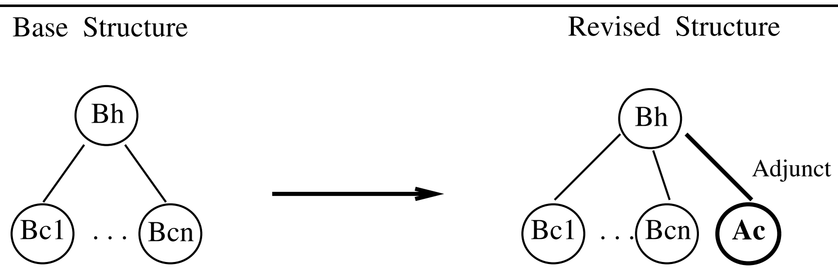

Figure 2.9: `Adjoin` schema

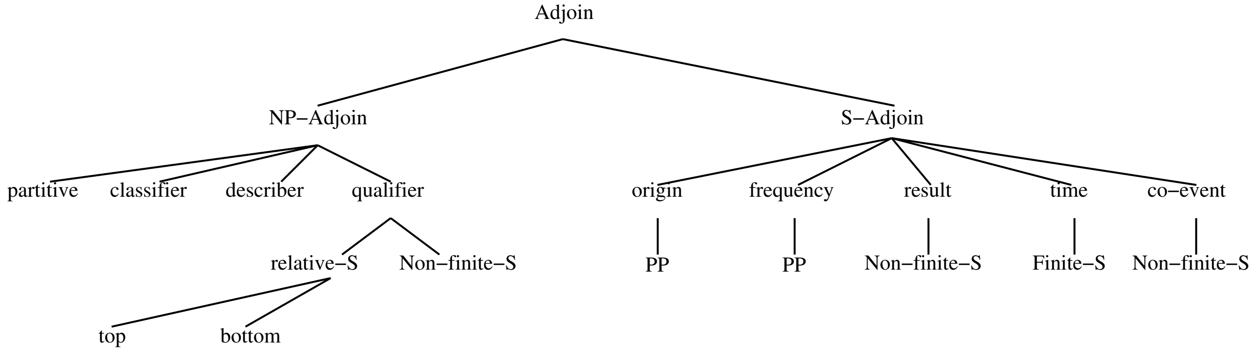

Figure 2.10: `Adjoin` revision operation hierarchy

Alternatively to the top-level nominal conveying the game result (underlined) as a whole and yielding:
"to power the Golden State Warriors to [[a 135 119 triumph over Los Angeles] , [**that extended the Clippers' losing streak to nine games.**]]"

The first revision is thus called `Adjoin of Relative Clause to Bottom-Level Nominal` and the second `Adjoin of Relative Clause to Top-Level Nominal`.

When adjoined to a clause, the new constituent could fill the following syntactic functions: frequency, result, time and co-event, with only a single syntactic category for the adjoined constituent in each case. `Adjoin` was used to add both types of historical information – streaks and records – as well as non-historical information.

The pair of phrases below illustrates how `Adjoin` can be used to add a record information to a game statistic nominal:
`Base phrase`: "Armon Gilliam scored [39 points]"
`Revised phrase`: "Armon Gilliam scored [**a franchise record** 39 points]"

The noun compound "franchise record" (corresponding to A_c in the schema) is simply added as a pre-modifying classifier of the "39 points" nominal (corresponding to B_c in the schema). This is a case of `Adjoin of Classifier`.

The surface decrement pair $< R_{i1}, R_{d1} >$ given Fig. 2.6 Section 2.5.1 provides an example of `Adjoin of Frequency PP to Clause`. In this example, the purpose PP realizing an additional streak fact (*e.g., "for its third straight win"*), is adjoined to the clausal base structure realizing a basic game result cluster (*e.g., "Chicago defeated Phoenix 99-91"* resulting in the revised cluster "*Chicago defeated Phoenix 99-91 for its third straight win*").

Note how in both these examples the linguistic expression of the base phrase content is not affected by the revision. The three other types of monotonic revisions, `Conjoin, Append` and `Absorb` are presented in Appendix A.

2.5.2.2 Non-monotonic revisions

I identified five main classes of non-monotonic revisions: `Nominalization, Adjunctization, Recast, Argument Demotion` and `Coordination promotion`. Each type is characterized by a different set of restructuring transformations which involve displacing base constituents, altering the base argument structure or changing the base lexical head. These types of transformations also distinguish non-monotonic revisions (as a whole) from monotonic revisions. Monotonic revisions conserve both the argument structure and the lexical head of the base and do not involve moving base constituents around.

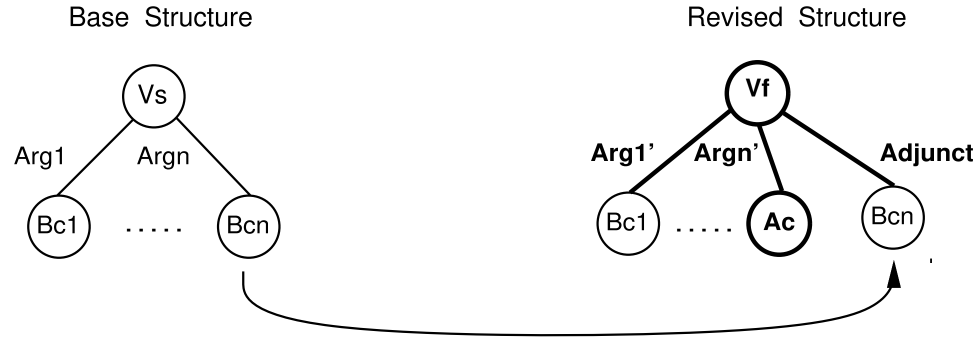

Figure 2.11: `Adjunctization` schema

Non-monotonic revision tools tend to be less versatile than monotonic ones and be applicable only to specific types of base structure. For example, `Adjunctization` applies only to clausal base patterns headed by a support verb V_s. Following Gross [Gross 1984], I call *support verb* any verb that does not realize any semantic element. Appearing only because each clause syntactically requires a verb in English, its sole function is to support one of its meaning-bearing arguments. Support verbs are opposed to *full verbs*, which do convey a semantic element by themselves. A support verb can have an argument set of various length. The supported argument can fill various semantic roles but it distinguishes itself from the other arguments in that it must be realized by an NP whose head collocates with the support verb. In the corpus sentence examples of the realization pattern R_{d2} given in Fig. 2.6 of Section 2.4.2, these collocations are verb-object collocations (*e.g.*, <*"to record"*, *"win"*>). The head verb (*e.g., "to record"*) does not realize any semantic element but only syntactically supports its range role, the NP (*e.g., "a 98-87 win against Dallas"*) whose head noun (*e.g.,"win"*) realizes the semantic head of the game result fact.

The general `Adjunctization` schema is given in Fig. 2.11. The additional content is realized by a combination of two new constituents: a full-verb V_f (i.e. a verb that bears meaning on its own) and its new object A_c. Deprived of the head verb that supported it in the base clause, original object B_{c_n} migrates to an adjunct position in the revised clause. It has thus been *adjunctized*.

The pair of phrases below illustrates how `Adjunctization` can be used to add a streak interruption to a game result clause:
`Source phrase`: "the Denver Nuggets *claimed* A 124 110 VICTORY OVER THE DALLAS MAVERICKS"
`Target phrase`: "the Denver Nuggets **ended their three game losing streak with** A 124 110 VICTORY OVER THE DALLAS MAVERICKS"

The streak information is introduced by a new full-verb "to end" (corresponding to V_f in the schema) and a new object nominal "their three game losing streak" (corresponding to A_c in the schema). Deprived of its support verb "to claim" (corresponding to V_s in the schema), the original object nominal "a 124 1110 victory over the Dallas Mavericks" (corresponding to B_c in the schema) conveying the game result migrates as an instrument adjunct. Since the thematic role of the original object in the source phrase was range, this is a case of `Adjunctization of Range Argument into Instrument`.

The surface decrement pair $< R_{d2}, R_{i2} >$ of Fig. 2.6 in Section 2.5.1 provides another example application of this revision tool, this time to add a streak extension. Note how in both examples the support verb of the base pattern is deleted by the revision.

The other types of `Adjunctization` are shown in Fig. 2.12. Cases of `Adjunctization` are first characterized by the target adjunct role of the displaced constituent. In the corpora analyzed, there were three such target roles: `instrument, opposition` and `destination`. Constituents moved to the destination role were all originally filling the `created` role and all those moved to the opposition role were originally filling the

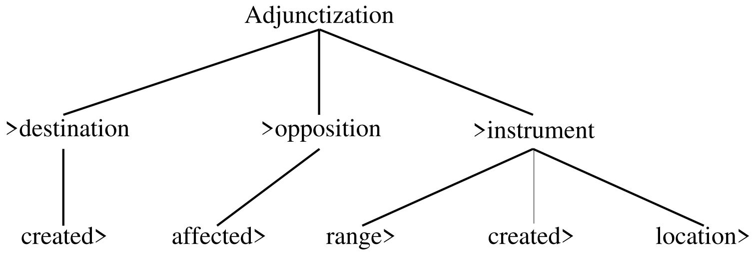

Figure 2.12: Adjunctization revision operation hierarchy

`affected` role. Those moved to the instrument role were coming from either `created`, `range` or `location` role positions.

The four other types of non-monotonic revisions, **`Nominalization`**, **`Recast`**, **`Argument Demotion`** and **`Coordination promotion`** are presented in Appendix A.

2.5.2.3 Side transformations

The previous subsection presented the *core* transformations involved in the corpus revisions:

- The introductory transformation whose goal is to attach the additional fact to the base pattern.
- The restructuring transformations whose goal is to maintain grammaticality.

Some revisions, however also involve *side* transformations that satisfy other goals such as avoiding repetitions, avoiding ambiguities and enforcing collocation constraints. I identified five main classes of side transformations: *Reference Abridging, Ellipsis, Argument Control, Scope Marking, Lexical Adjustment* and *Ordering Adjustment.*

The most common of these transformations is reference abridging. Its goal is to suppress repetitions introduced by the attachment of the additional fact. In the corpus, initial references use a set of default properties associated with the class of the referred entity in the domain ontology. For example, initial references to players use their first and last name, initial references to teams use their name and location, etc. However, when a revision introduces a second reference, this set of properties is *distributed* between these two references. The base reference is thus *abridged* by the revision[13].

The surface decrement pair $< R_{d3}, R_{i3} >$ below illustrates this phenomenon:

R_{b3} "to direct the Los Angeles Lakers past [*the* Washington *Bullets*] 87 72"

R_{r3} "to direct the Los Angeles Lakers past [Washington] 87 72 **handing** *the Bullets* **their ninth straight defeat**"

In the base pattern Rb3, both the name, "*the Bullets*" and the location, "*Washington*" of the losing team are used in the single reference to that team. When this cluster is revised, the added constituent contains a second reference to this team. In this second reference the team name alone is used. As a side transformation, the first reference is abridged using the team location alone. Without such a side transformation, the team name would be repeated just three words after being introduced:

$R_{r3'}$ "to direct the Los Angeles Lakers past the Washington Bullets 87 72 **handing the Bullets their ninth straight defeat**"

[13]Note that the use of this transformation is based on the assumption that the targeted audience of the report knows about every property in the default set, since otherwise it could not establish the co-reference link in the revised pattern

The other types of side transformations are presented in Appendix A.

2.5.2.4 Classifying criteria

Table 2.1 summarizes the essential characteristics of the revision tools I identified in the corpus. The columns of this table provide five essential criteria to classify these tools:

- The type of base pattern structures on which they can be applied: whether a tool is applicable only to hypotactic bases, only to paratactic bases or equally well to both.
- The type of revised pattern structures resulting from their application: either paratactic or hypotactic and in this latter case whether the revised pattern head is the base pattern head or the new constituent introduced by the revision.
- How base pattern constituents are displaced by the revision.
- How the argument structure of the base pattern is affected by the revision, whether it is expanded, shrunk, etc.
- How the lexical head of the base pattern is affected by the revision and in particular whether the revision involves replacing a support verb by a full verb or vice-versa.

Each distinct combination of these five factors defines a main class of revision. Each instance of revision can be further characterized by taking into account the following factors:

- The linguistic rank of the revision base pattern[14]: clause, nominal etc.
- The syntactic category of the constituent added by the revision.
- The grammatical function filled by this added constituent in the revised cluster.

The eight criteria above classify revisions in terms of core transformations. For revisions that also involve side transformation(s), the various types of side transformations constitute an additional classifying criterion. The resulting hierarchy, whose top-level was given in Fig. 2.8 and whose `adjoin` and `Adjunctization` subtrees where respectively given in Fig. 2.10 and Fig. 2.12, is fully presented in Appendix A.

2.5.2.5 Constraints on revision tools application

Table 2.2 summarizes five different constraints on the application of each of the ten basic revision tools. The first column specifies the types of side transformations that may accompany their application and the second column the linguistic ranks of the base patterns onto which they are applied. The final three columns specify the types of facts they can add: streak, records and/or box-score facts (i.e. non-historical information). Table 2.2 also contains occurrences of each of these constraints in the corpus of 270 surface decrement pairs. For example, concerning the monotonic revision tool `Adjoin` presented in Section 2.5.2.1, table 2.2 indicates that there were 88 total occurrences of its usage in the corpus, breaking down into 72 occurrences at the nominal-rank and 16 at the clause-rank. It also notes that among the 16 clause-rank occurrences, eight were used to convey a streak, one to convey a record and the seven remaining to convey a non-historical statistic. Finally, it indicates that 17 occurrences of `Adjoin` involved a side transformation, 15 reference adjustments and two ordering adjustments.

2.6 Summary

The corpus analysis described in this section has fulfilled five functions:

[14] The constraints on this rank for each basic tool type are given in table 2.2

	base structure	revised structure	subconstituent displacement	argument structure	head
adjoin	hypotaxis	hypotaxis head = base const	none	untouched	untouched
append	parataxis	parataxis	none	untouched	untouched
conjoin	any	parataxis	none	demoted	demoted
absorb	any	hypotaxis head = added const	none	demoted	demoted
recast	hypotaxis	hypotaxis head = base	argument $\mapsto$ adjunct	untouched	untouched
argument demotion	hypotaxis	hypotaxis head = base const	top-level arg $\mapsto$ embedded const	shrunk	untouched
nominalization	hypotaxis	hypotaxis head = added const	none	expanded	fV $\mapsto$ sV
adjunctization	hypotaxis	hypotaxis head = added const	argument $\mapsto$ adjunct	changed	changed
coordination promotion	hypotaxis w/ paratactic arg	parataxis	multiple	demoted	demoted

Table 2.1: Comparative structural characterization of the revision tools

		side transformations		scope ranks		streak info	records info	*non-* histo
adjoin	88	reference adjust	15	clause	16	8	1	7
		ordering adjust	2	nominal	72	23	46	3
append	10	ellipsis	1	clause	8	-	-	8
				nominal	2	-	-	2
conjoin	127	reference adjust	3	clause	91	5	-	86
		scope marking	12	NP coordination	28	-	-	28
		lexical adjust	3	NP apposition	8	8	-	—
		ellipsis	86				-	—
absorb	11	argument control	10	clause	10	3	-	7
				nominal	1	-	1	—
recast	4	none		clause	3	2	1	—
				nominal	1	1	-	—
argument demotion	1	none		clause	1	7	-	—
nominalization	1	none		clause	1	1	-	—
adjunctization	27	reference adjust	6	clause	27	15	8	4
coordination promotion	1	none		clause	1	-	1	—
	270		138		270	67	58	145

Table 2.2: Constraints on revision tools usage

- Identify the ontology of the basketball report domain.
- Identify the sentence-rank schemata used in the report leads.
- Identify realization patterns determining which syntactic structures can be used to express each concept combination allowed by these schemata.
- Identify revision tools to incrementally build complex realization patterns from basic ones.
- Identify constraints on the applicability of these revision tools.

The corpus analysis thus provided the data for all the knowledge sources of the prototype generator STREAK. It has also defined a target output which is useful to evaluate the success of STREAK. Finally, it has confirmed the plausibility of the revision approach by showing the highly compositional and regular structure of the complex sentences found in the corpus.

Chapter 3

A new generation architecture

In the introduction of this thesis I proposed a new draft and revision approach to language generation addressing the difficult issues raised by summarization applications. Implementing this new approach requires defining a new generation system architecture. The object of the present chapter is to motivate and present this architecture in detail.

Language generation is a very complex process involving a large and heterogeneous set of tasks. The architecture of a generation system specifies:

- The decomposition of the system into components, each responsible for a specific set of subtasks.
- The knowledge sources accessible to each component.
- The type of internal representations exchanged among components.
- The thread of control between components.

The architecture presented in this chapter is for a complete report summary generation system, producing multi-sentential text from raw quantitative data. As much as possible, it is presented independently of any specific implementation. One specific implementation of the most original parts of this architecture, the system STREAK, is presented in the next two chapters.

No standard terminology has emerged for the various subtasks of text generation. Different researchers have used the same terms to mean different things in the literature. This has obscured the issues. To avoid this pitfall, this chapter starts by defining a terminology for discussing architectural issues that is then used throughout the thesis. I then motivate the new architecture I propose in two steps. I start from a set of *observations* resulting from the corpus analysis of human-written summaries presented in the previous chapter. From these observations, I identify a set of *abilities* needed by a generator to produce similar texts. I then derive a set of *design principles* which provide a generator with these abilities. Finally, I present in detail the proposed architecture based on these principles, explicitly stating where and how each generation subtask defined gets carried out.

3.1 Text generation subtasks

One problem in discussing architectural issues in generation is the absence of a precise and standard terminology: the same terms have been used to describe different notions by different authors in the literature. In this subsection I briefly define a vocabulary that I will then use throughout the paper.

Text generation is traditionally decomposed in three subtasks:

- *Content determination*, answering the question "what to say?"
- *Content organization*, answering the question "when/where to say what?"

- *Content realization*, answering the question "how to say it?"

Together, the first two subtasks are also generally referred to as *content planning*, *deep generation* or *strategical generation*. The third subtask has been alternatively called *language synthesis*, *surface realization*, *surface generation* or *tactical generation*. Useful as they are, these threefold or twofold decompositions are too coarse grained. Each of these three tasks is in itself very complex. To be able to discuss how these two/three subtasks are carried out and compare what they recover in different generation architectures, it is necessary to make additional distinctions that further decompose the text generation process.

Content determination can be itself decomposed into content *production* and content *selection*. Although in many cases an underlying application provides the generator with all the potential content, in other cases the generator's input is but one part of that content. The rest of the content has to be produced by the generator itself. This is what Hovy calls interpretation in generation [Hovy 1988]: the generator needs to enrich its input with more content. In the extreme case the input consists only of communicative goals and it is the generator's task to produce all content from the knowledge sources it can access. Another case requiring content production by the generator is when its input format is totally inadequate for content planning. The generator must first produce content of an acceptable format before being able to start planning. This is the case for the type of generation application discussed in this thesis. The raw quantitative data input to the generator in tabular form must first be converted to a symbolic form, which captures conceptual generalizations and on which domain reasoning can be performed. This task of content *production* subtly differs from content *selection* which consists of deciding which part of the produced content is to be included in the generated text.

What each generation subtask covers depends on the level of *microcoding* at which generation is performed. A *macro*coded generator like ANA [Kukich 1983] produces sentences by assembling entire clauses and group patterns *stored* as a whole in its phrasal lexicon. In contrast, a *micro*coded generator like EPICURE [Dale 1992] dynamically builds from a word-based lexicon[1] every sentence constituent down to the group rank. The level of microcoding is defined by the minimal linguistic rank of the entries stored in the generator's lexicon. It is a crucial architectural characteristic because, apart from content realization, it also has repercussions on content selection and content organization. Both tasks significantly differ in nature depending on the rank of the linguistic unit being planned: text, paragraph, sentence, clause or group. In particular, macrocoded generators are *not* concerned with content selection and organization at the clause and group ranks. For these systems, everything below the sentence level is a realization matter. In contrast, for microcoded generators, the distinction between content selection, content organization and content realization is relevant even at the clause and group ranks. Thus, while for ANA generating *"The industrial average"* instead of *"The Dow Jones average of 30 industrials"* is a realization decision, for EPICURE generating *"the ripe banana"* instead of *"the banana"* results from content selection and organization choices. Generating *"the large shrimp"* and not *"the big prawn"* would be a genuine realization choice for EPICURE. In the rest of the paper, I therefore distinguish between *discourse level* content selection and organization at the text and paragraph ranks and *phrase level* content selection and organization at the sentence, clause and group ranks. One of the most innovative characteristics of the generation model proposed in this thesis is that it allows micro-level content selection and organization to be performed under surface form constraints.

Content realization can be decomposed into the four following subtasks:

- *Lexicalization*: the expression of semantic content by choice of open-class lexical items.
- *Semantic grammaticalization*: the expression of semantic content by choice of grammatical category (e.g., clause vs. NP) and grammatical features (e.g., tense for clauses, definiteness for NPs).
- *Morpho-syntactic grammaticalization*: the enforcement of syntax by choice of closed-class lexical items, choice of open-class lexical item inflections, specification of constituent precedence constraints etc.
- *Linearization*: the spelling out of the inflected lexical items following the precedence constraints.

It is during the first two of these subtasks that the mapping between content units and linguistic units occurs; everything preceding them is purely conceptual manipulation, while everything following them purely

[1] i.e. a lexicon whose entries are individual words

syntactic manipulation. Because they bridge the gap between conceptual and syntactic processing these two tasks have been considered part of content planning by some authors while part of content realization by others. I hold this second view.

The overall text generation task can thus be decomposed as follows:

- Content determination
  - Content production
  - Content selection
    - Discourse level content selection
    - Phrase level content selection
- Content organization
  - Discourse level content organization
  - Phrase level content organization
- Content realization
  - Lexicalization
  - Grammaticalization
    - Semantic grammaticalization
    - Morpho-syntactic grammaticalization
  - Linearization

The present thesis focuses on the following generation subtasks: phrase level content selection, phrase level content organization and all four content realization subtasks.

3.2 Motivation for a new architecture

In the introduction of this thesis I presented the difficult issues that summarization applications raise for a language generator: conciseness, sentence complexity, floating concepts, historical background and paraphrasing power. In this subsection, I propose four principles for designing a generation system that handles these difficult issues: (1) incremental draft and revision approach, (2) microcoding from a word-based lexicon, (3) presence in the draft representation of a purely conceptual layer independent of linguistic form and (4) presence in the draft representation of a surface layer reflecting realization choices. These principles are essentially motivated by a set of corpus observations on the way historical information is conveyed in human-generated reports. From these observations I deduce a set of abilities that a generation system must have in order to concisely and flexibly convey historical information. These abilities can be provided by the four design principles above.

3.2.1 From corpus observations to needed abilities

The corpus observations which motivate the need for a new generation architecture are the following:

1. Historical information is combined with new information in sentences of a greater complexity than those produced by existing generation systems.
2. Historical information of the same type is conveyed by a wide range of different syntactic constructs.
3. Historical information of the same type is conveyed at different linguistic ranks.
4. Historical information of the same type is scattered in distant locations inside a report.

Variety of syntactic constructs to convey the same additional historical fact:

- `Clause-complex coordinative conjoin`:
  (1) "David Robinson scored 32 points Friday night LIFTING THE SAN ANTONIO SPURS TO A 127 111 VICTORY OVER DENVER **and handing the Nuggets their seventh straight loss**".
- `Adjoin of non-finite clause in top-level nominal`:
  (2) "David Robinson scored 32 points Friday night lifting the San Antonio Spurs to A 127 111 VICTORY OVER DENVER **sending the Nuggets to their seventh straight loss**".
- `Adjoin of relative clause in top-level nominal`:
  (3) "David Robinson scored 32 points Friday night lifting the San Antonio Spurs TO A 127 111 VICTORY OVER DENVER **that extended the Nuggets' losing streak to seven games**".
- `Adjoin of relative clause in embedded nominal`:
  (4) "David Robinson scored 32 points Friday night lifting the San Antonio Spurs to a 127 111 victory over THE DENVER NUGGETS **who lost for the seventh consecutive time**".
- `Top-level nominal appositive conjoin`:
  (5) "David Robinson scored 32 points Friday night lifting the San Antonio Spurs to A 127 111 VICTORY OVER DENVER, **the Nuggets' seventh straight defeat**".
- `Embedded nominal appositive conjoin`:
  (6) "David Robinson scored 32 points Friday night lifting the San Antonio Spurs to a 127 111 victory over THE DENVER NUGGETS, **losers of seven in a row**".

Figure 3.1: Syntactic diversity of historical information

5. Taking into account historical information triggers a combinatorial explosion of the number of relevant facts to consider for inclusion in the report.
6. Assuming a draft and revision approach, the applicability of revision tools for adding an historical fact on a given draft sentence is constrained by the surface form of that sentence.

In the following paragraphs I review in turn each of these observations and their consequences in term of desired abilities for a report generator conveying historical information.

The first three of these observations have already been discussed in the introduction of this thesis. The first observation, the syntactic complexity associated with historical information, was summarized in the table of Section 1.1.2.4 comparing corpus sentences containing historical information with sentences produced by existing generators in terms of number of words and facts. This table showed that the ability to convey historical information entails *the ability to generate very complex sentences.*

The second observation, the syntactic variety associated with historical information, is illustrated in Fig. 3.1, first presented in the introduction on p.5 and duplicated here. This figure contains six corpus paraphrases, containing an historical streak fact emphasized by a boldface font. In each of these paraphrases, the same historical fact is attached to the same basic sentence by a different syntactic construct. The ability to flexibly convey historical information thus entails *the ability to generate a wide variety of syntactic constructs below the sentence rank.*

The third and fourth observations are based on the notion of "floating semantic element". A fact to convey in a report is a "floating semantic element" if it can alternatively be realized at various levels inside the report structure. *Floating* semantic elements are opposed to *fixed* semantic elements which are always realized at the same given level. In Section 2.4, I noted that in the corpus reports, the main statistic and the game result were systematically realized by the two top-level clauses of the first sentence. These two facts are thus examples of fixed semantic elements. There are two aspects to the phenomenon of floating semantic element: an *intra*-sentential aspect and a *inter*-sentential aspect. Intra-sentential floating refers to the property of a fact to be alternatively realizable at different linguistic ranks inside a given report sentence. Inter-sentential floating refers to the property of a fact to be alternatively realizable in different and perhaps distant report sentences. In the corpus, historical information displayed both characteristics of floating information[2].

The six paraphrases of Fig. 3.1 illustrate intra-sentential floating of an historical fact. In these paraphrases, the same historical fact is realized at different linguistic ranks: clause-complex in (1), clause in (2), (3) and (4), nominal in (5) and (6). This example shows that the ability to flexibly convey historical information entails *the ability to realize floating semantic elements at different linguistic ranks.*

The corpus report of Fig. 3.2, shows how an historical fact can float across the entire report structure. Consider where the streak fact (emphasized by a boldface font) about each team is conveyed. Denver's streak is attached to the game result in the *first sentence* of the report. In contrast, Sacramento's streak is attached to the scoring statistic of one of its players in the *last paragraph.* It is essential to note that the distance separating the two streak facts in this report, cannot be explained on semantic or syntactic grounds. The only semantic constraint on the attachment of a team's streak to a draft sentence is that this sentence contains a reference to that team. The first two sentences of the report contain a reference to Sacramento (emphasized by a smallcap font). Semantically, they are thus as valid a location for Sacramento's streak as the last paragraph's first sentence. Syntactically, they are also valid locations since the relative clause which realizes Sacramento's streak in the last paragraph could also have been added as modifier of either reference, as shown by sentences S_1 and S_2 below.

S_1 "Sacramento, Ca. – Michael Adams scored a career-high 44 points Wednesday night, including seven 3-point baskets, to help the short-handed Denver Nuggets end a five-game losing streak with a 128-112 victory over THE SACRAMENTO KINGS, **who lost their fourth straight**".

S_2 "Adams, who was drafted and then discarded by THE KINGS, **who lost their fourth straight**, four seasons ago, made 17 of 26 field goals, including seven of 11 3-point attempts, and hit three of four free throws to break his previous career high of 35 points."

The preference of the first sentence of the last paragraph over S_1 can be explained on *discursive* grounds. Recall from Section 2.2 that because the corpus reports follow an inverted pyramid structure with summary lead, only crucial facts go in the first sentence. Sacramento's streak information was probably not important enough to have been incorporated in the first sentence. As for the preference of the first sentence of the last paragraph over S_2, it can be explained on *stylistic* grounds. The embedding of the relative clause realizing the streak inside another relative clause, makes S_2 stylistically awkward. The example report of Fig. 3.2 thus shows that choosing between alternative report structure locations for realizing floating historical facts requires taking into account not only semantic and syntactic factors but also discursive and stylistic ones. The ability to flexibly convey historical information therefore entails *the ability to realize floating semantic elements under a combination of semantic, syntactic, stylistic and discursive constraints.*

The fifth observation concerning historical information in report generation is that it triggers a combinatorial explosion of the number of relevant facts. This explosion is due to the fact that the historical context multiplies the dimensions along which to evaluate the significance of each input statistic. Consider for example, a player's scoring statistic in a basketball game. Ignoring the historical context there are basically only two ways in which this statistic can be significant: when compared to the scoring statistics of all the other players in the game (game-high) and when compared to the scoring statistics of his teammates only (team-high). However such a statistic can be *historically* significant in a combinatorially explosive number of ways: when compared to that player's scoring average (or high or low) over the season (or over his entire career or since he joined his current team, etc), when compared to the highest scoring performance of any

[2]The fact that in the corpus, historical semantic elements of given type are coerced to appear in one cluster of the lead sentence and not the other, does not invalidate the floating nature of these elements. The clusters' internal complexity allows these elements to appear at different linguistic ranks inside a given cluster.

Sacramento, Ca. – Michael Adams scored a career-high 44 points Wednesday night, including seven 3-point baskets, to help the short-handed Denver Nuggets **end a five-game losing streak** with a 128-112 victory over THE SACRAMENTO KINGS.
Adams, who was drafted and then discarded by THE KINGS four seasons ago, made 17 of 26 field goals, including seven of 11 3-point attempts, and hit three of four free throws to break his previous career high of 35 points. Adams also dished out a game-high 10 assists and had five steals.
Rookie Chris Jackson added 22 points and center Blair Rasmussen pumped in 21 and grabbed 12 rebounds for Denver, which outscored Sacramento 19-4 during the final 5:01 of the fourth quarter.
The Nuggets, who had only eight players available for the game, improved to 2-12 on the road and 6-20 overall.
Rookie guard Travis Mays, playing his second game after missing 11 with back spasms, scored a season-high 36 points for THE KINGS, **who lost their fourth straight**. Lionel Simmons added 24 points and 15 rebounds.

Figure 3.2: Streak information in a corpus report

player on his team (or on any team in a given division, conference, etc) this season (or over the last 10 games, or ever) while playing at home (or on the road) against a particular opponent (or against any opponent) etc. Because of the sheer number of historical facts, any such fact is likely to be of similar relevance as many others. Therefore, content production results in a much larger set of candidate facts with many more shades of relative importance. This makes content selection much harder. General rules based on purely encyclopedic grounds (*e.g.,* if a player scores more than 20 points or if he scores more than anybody in his team, then include his scoring in the report) no longer suffice to decide which facts to select. Other factors need to be considered. One such factor is whether the realization patterns of the candidate facts can be combined in cohesive and stylistically felicitous surface forms. The ability to convey historical information thus entails the ability *to perform final content selection under surface form constraints.* For example, to decide not to include a complementary fact because incorporating it would require either generating too complex a sentence or too long a summary.

The sixth and last corpus observation concerning historical information is that which type of revision can be applied to a report draft to add a given historical fact is constrained by the surface form of that draft. This observation has already been made in Section 2.5.2. The first column of table 2.1 in Section 2.5.2 indicates that the applicability of most revision tools is restricted to base clusters with specific syntactic structures. Consider again the basic corpus sentence example given in the introduction:
(0) *"David Robinson scored 32 points Friday night lifting the San Antonio Spurs to a 127 111 victory over the Denver Nuggets"*.

The different revisions of this base sentence to add the same additional streak fact shown in Fig. 3.1 all result from using two revision tools: conjoin and adjoin[3]. Other revision tools, cannot be used with (0) due to its surface form. For example adjunctization[4], which requires the game cluster of the base sentence to be headed by a support verb, cannot be used with (0) which is headed by the full verb "to lift". Adjunctization, can, however, be used to add the same streak information on the base sentence (0') synonymous with (0), yielding (1') synonymous with the sentences (1-6) of Fig. 3.1:
(0') "*David Robinson scored 32 points Friday night and the San Antonio Spurs rolled to* `a 127 111 victory over the Denver Nuggets`"
(1') "*David Robinson scored 32 points Friday night and the San Antonio Spurs* **extended Denver's losing streak to seven games with** `a 127 111 victory over the Denver Nuggets`".

This example shows that, in some cases, the base sentence syntactic structure can be the only discrimi-

[3]The variety comes from different application of these tools (i.e. at different linguistic ranks and in different syntactic forms.) in each paraphrase.
[4]Presented in Section 2.5.2.

natory factor justifying the choice of one revision tool over another.

The conclusion drawn from the corpus observations listed in the beginning of this subsection is that the ability to convey historical information entails the following abilities:

1. Generating very complex sentences.
2. Generating a wide variety of syntactic constructs below the sentence rank.
3. Performing final content selection under surface form constraints.
4. Realizing floating semantic elements at different linguistic ranks.
5. Realizing floating semantic elements under a combination of semantic, syntactic, discursive and stylistic constraints.

3.2.2 From needed abilities to design principles

What architecture design principles can we deduce from the agenda of providing a report generator with the five abilities needed for flexibly convey historical information identified in the previous section?

In the introduction of this thesis I have already explained that generating a wide variety of syntactic constructs below the sentence rank requires microcoding sentences down to individual words (and collocations made of a very few words). From an engineering perspective, I have also suggested that the need to produce very complex sentences through microcoding calls for a two-pass draft and revision generation model.

The cognitive research literature also supports the hypothesis that to generate very complex sentences conveying many facts revision is needed. [Pavard 1985] describes an experiment providing psychological evidence supporting the revision model for the generation of very complex sentences. In this experiment, human subjects were asked to write a single sentence paraphrasing a text of three sentences which together conveyed eight facts in 42 words. They were thus asked to generate a sentence of complexity similar to the newswire report leads analyzed in Section 2 of this paper. The subjects were divided into two groups. In one group the subjects performed the exercise using a microphone, i.e. a medium that precludes revision, while in the other they performed the exercise using a text editor, i.e. a medium that facilitates revision. For each group two measurements where made:

- Average percentage of semantic elements omitted in the single-sentence paraphrases, among those realized by head and argument constituents in the original text.
- Average percentage of semantic elements omitted in the single-sentence paraphrases, among those realized by adjunct constituents in the original text.

The results of this experiment are summarized in the table below:

media	omitted heads and arguments	omitted adjuncts
microphone	4%	34%
text editor	0%	4%

These results strongly suggest that even humans have trouble generating sentences of such complexity in one shot. They also show that the trouble does not so much arise for fixed semantic elements (which essentially correspond to the elements realized by the head and argument constituents in the original text), but specifically for floating semantic elements (which essentially correspond to the elements realized by adjunct constituents in the original text). They thus support the cognitive plausibility of the generation model proposed here where floating elements are added by revising a draft built around fixed elements.

I have so far showed that the need for both syntactic variety below the sentence rank and syntactic complexity suggests two principles for a new report generation architecture: incremental information-adding revision and microcoding from a word-based lexicon. Within this framework, the three remaining abilities identified in the previous subsection (i.e. constraining final content selection by surface form factors, realizing floating semantic elements at different linguistic ranks and under a combination of semantic, syntactic, stylistic and discursive constraints) support two additional principles concerning the draft representation on which to perform revision: (1) that it include a surface layer reflecting realization choices and (2) that it include a purely conceptual layer independent of linguistic form.

In the draft and revision generation model I propose, final content selection occurs during revision. In particular, it is at that stage that the final decision to include a particular historical fact in the report is made. Within this framework, the ability to constrain final content selection on surface form factors requires the use of incremental revision to perform not only phrase planning but also lexicalization and semantic grammaticalization. The draft representation must therefore include a layer specifying the realization (*i.e.,* the open-class lexical items and syntactic form) of each fact. This need is also supported by the corpus observation that revision tool applicability is constrained by the surface form of the draft.

The need to handle inter-sentential floating of semantic elements provides an additional justification for both:

- The draft and revision approach sketched in the introduction of this thesis and elaborated in the next section
- The representation of realization choices in the draft.

At revision time, a generator has access to the draft of the *entire* report. If surface form is represented in the draft, the generator can then choose where to realize a floating semantic element under a combination of semantic, syntactic, stylistic and discursive constraints. In particular, it can choose between two candidate locations for realizing a floating semantic element by comparing the respective stylistic impacts of attaching the floating element to either of them, even if these locations are distant in the text structure. This would be impossible for a one-pass architecture where realization is performed linearly on a sentence-per-sentence basis. With such an architecture, when the current sentence is one of the potential locations for realizing a floating semantic element, there is no way to measure choosing this sentence versus other potential locations that may lie ahead in the text structure but for which no surface form is yet available.

The need to handle intra-sentential floating of semantic elements requires that the draft representation also include a purely conceptual layer that totally abstracts from linguistic form. The first constraint on the attachment of a floating semantic element onto a draft sentence constituent is semantic: the attachment must be warranted by some semantic relationship holding between the floating element and some base semantic element realized by the draft constituent. The same base element can be realized at different linguistic ranks in different sentences. For example, the base semantic element realized at the clause rank in phrase X below, is realized at the nominal rank in phrase Y:
(X) "*John Stockton scored 27 points*"
(Y) "*John Stockton's 27 points*"

Despite their syntactic differences, the same revision can be applied on both these phrases to incorporate a floating semantic element of type extrema, as shown by X' and Y' below:
(X') "*John Stockton scored* **a season-high** *27 points*"
(Y') "*John Stockton's* **season-high** *27 points*"

The revision component can determine that this revision is applicable to both these phrases, only if the draft representation captures the meaning identity between (X) and (Y). This identity can be captured only by a purely conceptual representation that totally abstracts from linguistic form. A linguistically motivated semantic representation such as PENMAN's Upper-Model [Bateman *et al.* 1990], would view (X) as a material action and (Y) as a possessed object, two radically different semantic categories. It would thus fail to capture their identity of meaning. This contrast between conceptual and linguistic semantic representations is further discussed in Section 3.12.

In this subsection I have shown that providing a report generator with the five abilities it needs to flexibly convey historical information suggest that its architectural design must reflect the four following principles:

1. Incremental information-adding revision
2. Microcoding from a word-based lexicon
3. Presence in the draft representation of a purely conceptual layer independent of linguistic form.
4. Presence in the draft representation of a surface layer reflecting realization choices.

In the next subsection, I present in detail a report generation architecture based on these four principles.

3.3 A revision-based architecture for incremental generation

In this subsection, I propose a new generation architecture based on the four principles defined in the previous section. This new architecture is a general design for any report generator handling historical information in a quantitative domain and it encompasses the full range of report generation subtasks[5]. This thesis focuses on three of these subtasks: phrase level content selection, phrase level content organization and content realization. These three subtasks have been implemented in the summary generation system STREAK. Chapter 4, where I present this implementation, contains the full details about these three tasks. Issues related to generation subtasks whose implementation lay beyond the scope of the research presented in this thesis, namely content production and discourse level planning and organization, are discussed only at a high level and in general terms.

I start by describing the various levels of representation of an utterance in the architecture. I then describe the various processing components of the architecture and how they interact to build a first draft of the report and then revise it. Exactly where each of the generation subtasks defined in Section 3.1 is carried out in this architecture is further discussed in the subsequent section.

3.3.1 Internal utterance representation

In the architecture I propose, an utterance is represented at three different levels of abstraction:

- the Deep Semantic Specification (DSS),
- the Surface Semantic Specification (SSS),
- the Deep Grammatical Specification (DGS).

The architecture thus belongs to the stratificational tradition of computational linguistics (cf. [Dale 1992] and [Polguere 1990] for other generation systems using a stratificational utterance representation scheme), with multiple representation layers, each capturing a specific set of regularities. The general stratificational scheme I propose is sketched in Fig. 3.3. It is based on two assumptions concerning the generation system. The first is the presence of an interface linking the generator to the underlying application program. This interface performs content production. In the case of summary report generation from quantitative data, the interface is a fact generator that retrieves interesting data from tables of numbers and reformats them as conceptual structures suitable for text generation. For other applications this interface may query a database, the trace of an expert system, an interlingua representation of a text to translate, etc. The second assumption is that the generator does not perform low-level syntactic processing on its own, but instead relies on a stand-alone, portable syntactic grammar of the target natural language such as SURGE [Elhadad 1993b], NIGEL [Mann and Matthiessen 1983] or MUMBLE [Meteer *et al.* 1987] for English. The three layers define a pipeline of internal representations that bridge the gap from the application program interface that performs domain-specific, language-independent, conceptual processing, to the syntactic grammar component that performs domain-independent, language-specific, linguistic processing. I describe and exemplify each layer in turn in the following subsections.

[5] These subtasks were defined in Section 3.1.

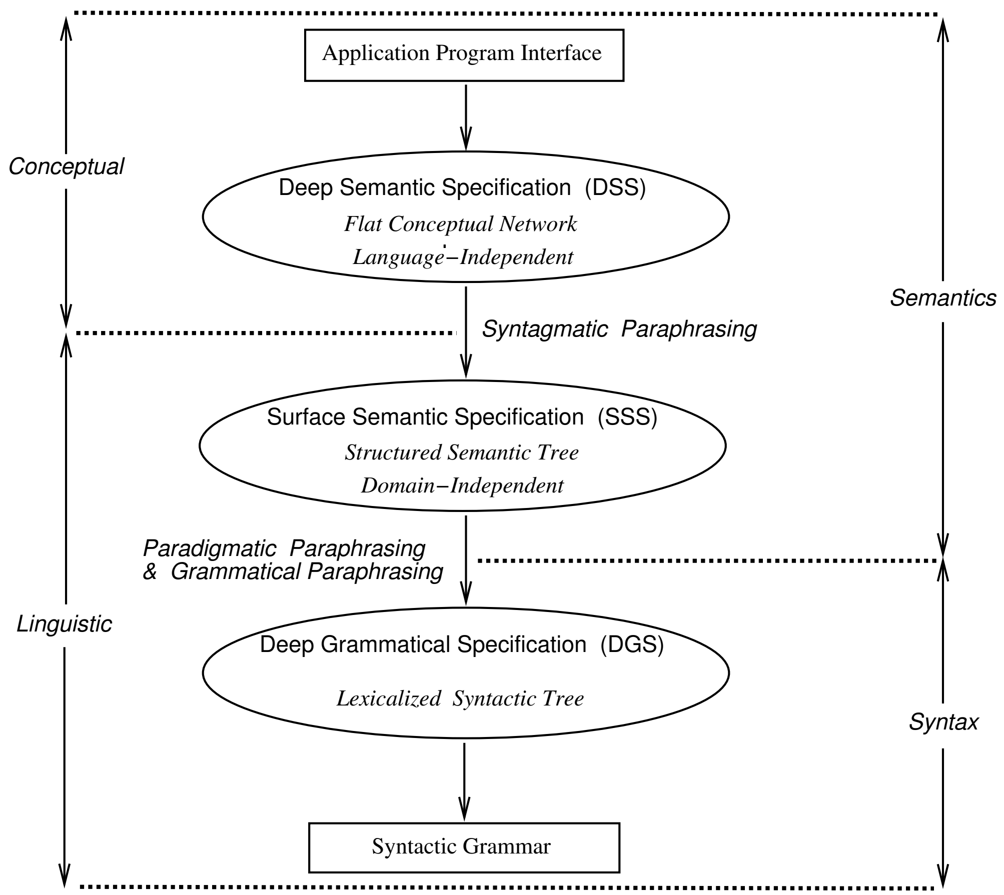

Figure 3.3: Internal representation layers

3.3.1.1 The Deep Semantic Specification

The Deep Semantic Specification is a flat conceptual network. It is a partial description of an event or an object of the underlying application domain like the kind that can be found in knowledge bases using a relational representation formalism. The nodes of the network are concepts and the arcs are role relations among them. An example DSS is given in Fig. 3.4. It represents the result of a basketball game between two teams, the Orlando Magic and the Toronto Raptors, specifying that the game was won both by and in Orlando. In such a DSS network, a concept is represented by a rectangle with its name prefixed by "c-" and role relation is represented by an oval with its name prefixed by "r-".

This example network illustrates the two key properties of a DSS network:

- *It explicitly contains redundant information* that can be deduced by domain reasoning from a minimal description. For example, in `dss0` both the `r-winner` and `r-loser` relations are present even though one could be deduced from the other by the common sense knowledge that team sports involve two teams, and that if one wins a game then the other necessarily loses it. Similarly, the relation `r-beat` between both teams is also redundant with `r-winner` and `r-loser`.
- *It is flat*, without any notion of head, constituent, up or down. Any node or arc in the network can potentially serve as head for a linguistic expression of the facts represented by the network.

These two properties insure that the DSS is totally uncommitted to any particular linguistic realization.

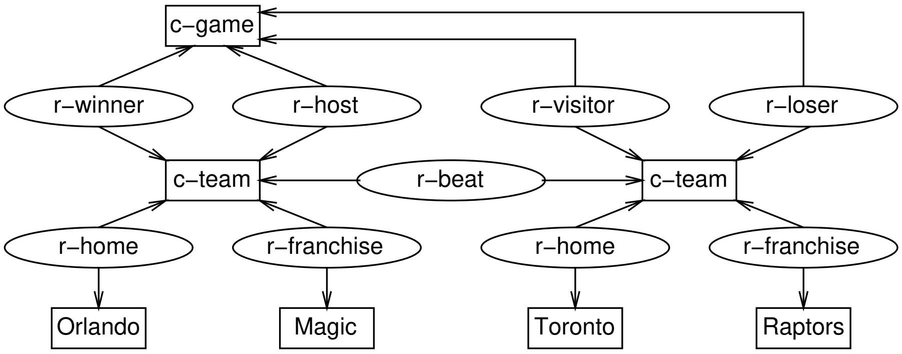

Figure 3.4: An example DSS: `dss0`

This makes the DSS an interface that cleanly separates conceptual domain reasoning from linguistic processing: the generator components upstream from the DSS are only concerned with domain reasoning and need not worry about linguistic considerations and conversely, the generator components downstream of the DSS are only concerned with linguistic processing and need not perform any domain reasoning.

Each redundant relation in the DSS captures a different *perspective* on the overall event represented by the DSS. Different paraphrases in the domain sublanguage describe the same event from different perspectives. Consider for example the following paraphrases describing the content represented by `dss0`:

1. "A game in which the Orlando Magic defeated the Toronto Raptors at home" describes the event from the perspective of the `r-beat` relation which is lexicalized by the verb "to defeat".
2. "The Orlando Magic won their home game against Toronto Raptors" describes the same event from the perspective of the `r-winner` relation which is lexicalized by the verb "to win".
3. "The Toronto Raptors lost the road game at the hand of the Orlando Magic" also describes the same event but from the perspective of the `r-loser` relation which is lexicalized by the verb "to lose".

Note that in each paraphrase a different concept is put in focus by appearing first in the sentence. This illustrates that such paraphrasing power is not only needed for the sake of variety (always using the same linguistic form immediately betrays the artificial nature of a generated text) but also to satisfy discourse constraints, such as focus shift rules [McKeown 1985] insuring that the sentence coherently inserts itself in the overall generated text. If only `r-winner` was present in the DSS, either (1) it would be impossible to put either the game itself or the losing team in focus as in sentences 2 and 3 above, or (2) the components downstream would themselves need to infer `r-beat` and `r-loser` from `r-winner`. With this last option the DSS would then fail to circumscribe domain reasoning to the application program interface.

This interface is thus responsible for providing in the DSS all the aspects of an entity that are described in any one of the domain sublanguage expressions referring to this type of entity. The linguistic components downstream of the DSS then decide which aspects to include explicitly in a given expression of the DSS content and which to leave implicit. This issue of implicit *vs.* explicit content realization is further discussed for example DSSs in the next section. It is in itself a vast issue that has been the object of several dissertations. In this thesis, I consider it only from the perspective of generating paraphrases in the context of written report production for restricted quantitative domains where the readership is assumed uniform. Therefore, I do not discuss its relation to issues such as user-modeling [Paris 1987], conversational implicatures [Reiter 1991] or argumentation [Elhadad 1993b]. For application domains where these issues are crucial, the DSS could not be assumed to include all the relevant aspects of an entity to describe. Instead, it would need to consist only of a default description sufficient for the most common situations. For the other situations, the linguistic component would need the ability to request more information to the application program interface whenever such information is needed to choose between a particular set of linguistic options. This issue of enriching the DSS on demand during linguistic realization and the implementation of a facility for such process within the generation framework of functional unification is discussed in detail in [Elhadad and Robin 1992]. During the implementation of the generator STREAK, I discovered that for the particular sports game summarization application of this system, this facility was not really needed and thus did not make use of it. Nevertheless I remain convinced that in other domains such a facility could be essential.

The approach to model the three paraphrases above taken in this thesis is to consider them as primarily resulting from different *conceptual perspective* choices. They do constrain lexical choices but are nonetheless kept separate. Perspective is chosen while mapping the DSS into an SSS. Only when mapping this SSS into an DGS, does a specific word get picked (among those whose argument structure is compatible with the chosen perspective). This approach also separates perspective choice from the domain reasoning necessary to specify the range of perspectives (this is done while building the DSS). This separation makes the implementation of each task much easier and also increases the potential for portability of each knowledge source. An alternative way to look at the three paraphrases above is to consider them as primarily resulting from different *lexical* choices, in the case at hand choosing among the verbs "*to defeat*", "*to win*" and "*to lose*". In terms of generator design such a view obviates the need for redundant information in the DSS. The three different words would then be considered alternative lexicalizations of a unique concept, for example `r-loser`. In order to generate sentence 3 above, this approach would require the lexicalizer to simultaneously:

- Reason that the host of a game whose visitor is also the loser must be the winner of that game.
- Choose `r-loser` as the head concept of the sentence but at the same time focus on the winning team.
- Choose "*to win*" as opposed to "*to crush*", "*to rout*", "*to whip*" and the hundred or so verbs which allow to realize `r-loser` while focusing on the winning team.

This alternative view thus has the disadvantage of burdening the lexicalizer component with domain reasoning and syntagmatic choices in addition to the paradigmatic choices which are its sole responsibility in the architecture proposed here. These paradigmatic choices can be subtle. For example, verbs such as "to crush" or "to whip" can be used only for a restricted set of final scores. Dealing with this issue separately from perspective and syntagmatic choices seems the best option.

The flatness of the DSS is just as important as its potential redundancy. Note, for example, the contrast between sentences (1) and (2) in the above paraphrase. The first is an NP whose head constituent realizes the `c-game` concept of `dss0`, while the second is a clause whose head constituent realized the `r-winner` relation of `dss0`. In many generators, the linguistic components are presented with tree-structured input in which the semantic element that will constitute the head of the linguistic expression is already implicitly chosen. Typically this input consists of a thematic (or case) role structure describing an action. Because these thematic roles are linguistic abstractions, there are constraints on which syntactic category can fill them. With such an input, the linguistic components therefore have very restricted freedom for choosing the position and category of each semantic element in the syntactic structure. They generally map the top level action into a clause following the input thematic structure and recursively map each role filler to an NP. If the DSS was already structured as a thematic role structure, either (1) each type of entity would always be described by the same thematic structure and syntactic category thus considerably limiting the paraphrasing power of the generator, or (2) the choice of thematic role structure would fall back to the application program interface. With this last option the DSS would then fail to shield this interface from linguistic considerations.

In short, the DSS must be a flat, partially redundant conceptual network whose function is to abstract from linguistic form by capturing the common recoverable meaning shared the all synonymous phrases in a given sublanguage.

3.3.1.2 The Surface Semantic Specification

The Surface Semantic Specification is a semantic tree. It still purely semantic in the sense that it does not specify any particular syntactic category or lexical item for the utterance to generate. However, its tree structure captures the organization of content inside a given class of linguistic structure. It specifies how semantic elements are to be grouped together inside linguistic constituents and which element should head the group. In addition to specifying constituency and dependency, an SSS tree also differs from a DSS network in that it contains only those semantic elements to realize explicitly in the linguistic structure. Finally, in contrast to the DSS that is domain-specific and language independent, the SSS is linguistically motivated and domain-independent[6].

The arcs of an SSS tree are general linguistically motivated rhetorical relations and thematic roles. There are two different types of nodes in an SSS tree:

- Encyclopedic nodes realizing a concept or a relation of the corresponding DSS viewed as a member of a particular ontological class.
- Rhetorical nodes which do not realize any particular element of the corresponding DSS but which instead function as an aggregation medium to group several such elements within a linguistic constituent.

The top level ontological classes used as perspective through which to present a semantic element are: `event`, `individual`, `set`, `place`, `time`, `quality` and `quantity`. They act as pre-selectors of the syntactic category that will ultimately be used to express the semantic element. By default, when an element is viewed as an event it will be expressed by a clause, when viewed as a individual or a set it will be expressed by an NP, when viewed by a place or time by a PP, when viewed as a quality as an adjective or adverb etc.

There are three types of rhetorical aggregates:

- *Hypotactic complexes*, with a head and dependents all realizing some semantic element in the corresponding DSS but with these elements not coming from a neatly delimited subnetwork in the DSS. Complex clauses with adverbial adjuncts are all represented by hypotactic complex at the SSS layer.
- *Paratactic complexes*, without a head but in which all elements share the same structural status. Conjunctions and appositions are represented by paratactic complexes at the SSS layer.
- *Rhetorical event structures*, with a head which does not in itself realize any element in the corresponding DSS but only aggregates subconstituents who do. Clauses headed by support verbs are all represented by such rhetorical events at the SSS layer.

Figures 3.5 and 3.8 show three examples of SSS trees. Each tree corresponds to a different class of syntagmatic paraphrases of the content represented by `dss0`. In these figures, encyclopedic nodes are prefixed by "E" and rhetorical ones by "R" (as in the DSS, atomic elements have no prefix).

`sss1` is a linguistic structure plan where the `r-beat` concept is chosen as the head with an event perspective. It views the situation described in `dss0` as the wining team being an agent whose action affects the losing team and occurs in a location where the winning team is host. It also indicates that the `r-home` property of each team must used to refer it. The elements of `dss0` that are explicitly realized in `sss1` are indicated by a boldface font in Fig. 3.6. `sss1` represents synonym phrases such as:
"Orlando defeating Toronto at home"
"Orlando triumphed over Toronto in its building"

In contrast, in `sss2` it is the concept `r-winner` that is chosen as the head and with an individual perspective. It views the same situation described in `dss0` as an asset of the winning team, obtained at the expense of the losing team. The location where this object was obtained is itself seen as an individual. `sss2` also indicates that this time the name of each team should be used to refer to them. The elements of `dss0` that are explicitly realized in `sss2` are indicated by a boldface font in Fig. 3.7. `sss2` represents synonym phrases such as:
"The Magic's homecourt win against the Raptors"
"The home victory of the Magic against the Raptors"

[6] At least for the part of a sublanguage that can be considered domain independent. In very specialized technical sublanguages a non-negligible proportion of the encountered syntactic constructs may be idiosyncratic to the domain.

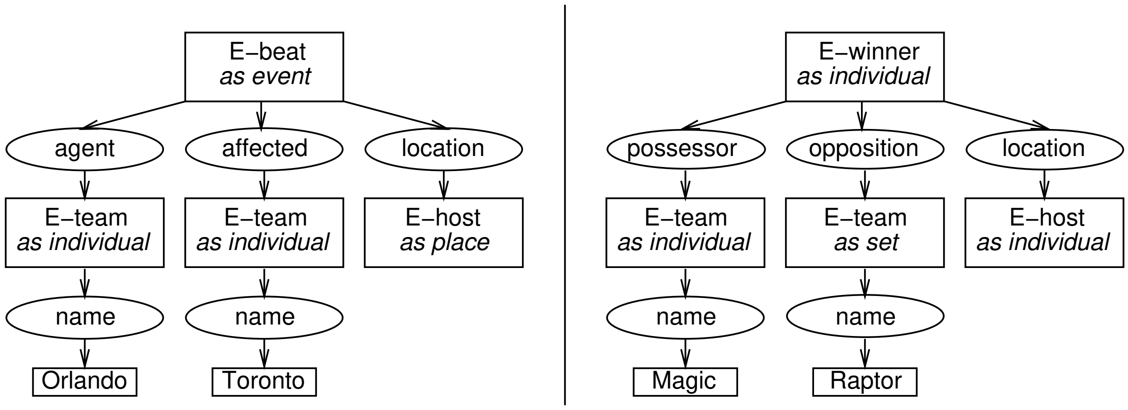

Figure 3.5: `sss1` (left) and `sss2` (right): two different perspectives on `dss0`

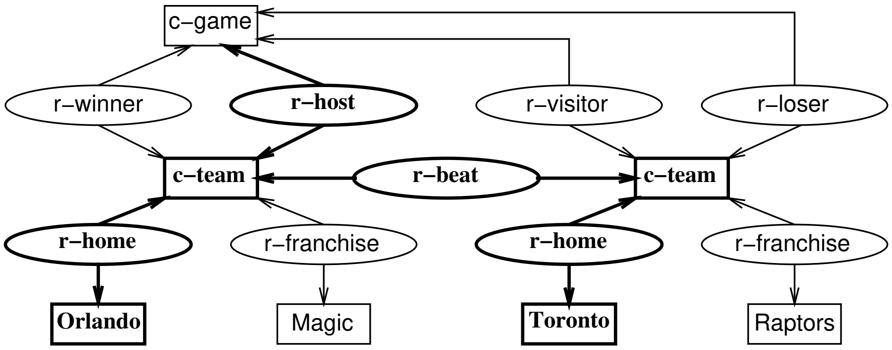

Figure 3.6: Content units of `dss0` explicitly realized in `sss1`

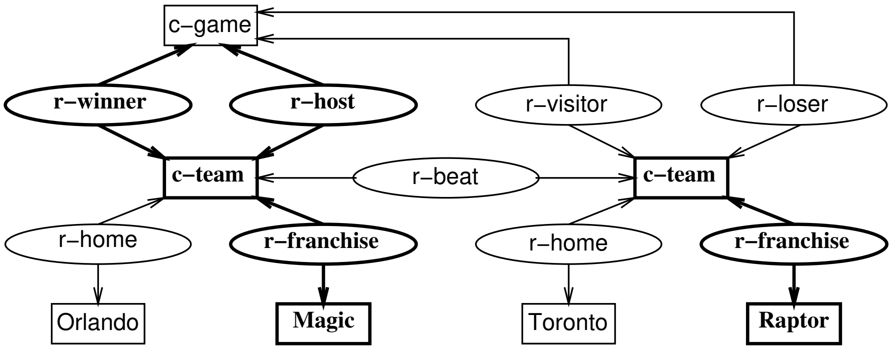

Figure 3.7: Content units of `dss0` explicitly realized in `sss2` and `sss3`

Finally, in sss3 none of the element from dss0 is chosen as head. Instead the head is a rhetorical node allowing the combining within an event structure of parts of the different perspectives respectively represented by sss1 and sss2. Like sss1, sss3 views the winning team as an agent, but it otherwise explicitly realizes the same elements of dss0 as sss2 and with the same perspective. It represents synonym phrases as:
"The Magic claimed a home triumph over the Raptors"
"The Magic posted a homecourt victory over the Raptors"

The contrast between these three example SSSs illustrates the one-to-many and non-isomorphic nature of the mapping from conceptual structure to linguistic structure even independently of lexical considerations.

3.3.1.3 The Deep Grammatical Specification

The Deep Grammatical Specification is a lexicalized syntactic tree. Each node corresponds to a syntactic constituent of the sentence to generate and specifies its category, lexical head and syntactic features (each category is associated with a set of relevant features, e.g., definiteness for NPs, mood for clauses). It is "deep" in the sense that (1) the arcs linking a clause to its subconstituents are still thematic roles like agent or affected instead of surface syntactic roles like subject or object and (2) only open-class words like verbs, nouns and adjectives are specified. Since the DGS is the input to a portable stand-alone syntactic grammar component it thus assumes that both the mapping from thematic roles to surface syntactic roles and the choice of closed-class words like articles and pronouns is carried out by this syntactic grammar from the syntactic features of the DGS[7]. The syntactic grammar also inflects open-class words while enforcing agreement, determines precedence constraints among constituents and words, and finally, outputs the stream of words satisfying these constraints.

The reason for the DGS to be structured in terms of thematic roles at the clause level is that the two most widely used portable syntactic grammars in generation research, today SURGE and NIGEL, are both based on the systemic linguistic framework [Halliday 1985], [Fawcett 1987] which include a semantic analysis of the clause in terms of a set of thematic roles. The advantage of such high level input to a syntactic grammar is to maximize the number of generation subtasks performed by this re-usable component. Its drawbacks are discussed in detail in Section B.4 of Appendix B. One is an asymmetry in the respective representations of internal structure of clauses and nominals. Systemic linguistic has not yet come up with a set of thematic roles for analyzing nominals similar to the set of roles it has developed to analyze clauses. The extreme semantic versatility of nominals makes their analysis in terms of thematic roles a daunting if intriguing task. In the current systemic framework, nominals are thus analyzed in terms of roles like describer, classifier and qualifier (cf. Section B.2 of Appendix B for their precise definition), which are more superficial that thematic roles and correspond more to the syntactic roles subject, object, complement, etc. of the clause.

The tree structure of the DGS is very similar to that of the corresponding SSS. The main difference occurs when an arc of the SSS is realized lexically in the sentence to generate. In this case, it corresponds to a *node* of the DGS instead of an arc. This difference is in part rooted in the asymmetry of the DGS layer for clauses and nominals just evoked. An example of such non-isomorphism between DGS and SSS is given on the left side of Fig. 3.9 showing, dgs21, the DGS lexicalizing sss2 by the phrase:
"The Magic's triumph over the Raptors".
The opposition arc in sss2 is realized lexically by the preposition "over" in dgs21, introducing an additional constituent.

Like the mapping from DSS to SSS, the mapping from SSS to DGS is one to many in addition to being isomorphic. For example, the right side of Fig. 3.9 shows dgs22, an alternative lexicalization of sss2 by the phrase:
"The Magic's homecourt win against the Raptors"

[7]Prepositions are the exception. Even though a closed class they cannot be chosen on purely syntactic grounds as shown by [Herskovits 1986]. I thus assume that the syntactic grammar can only provide a default choice which need to be overwritten in the DGS of utterances for which this default is not appropriate on semantic grounds.

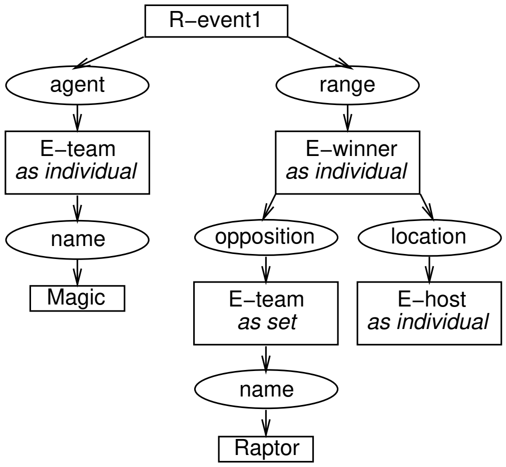

Figure 3.8: `sss3`: a third perspective on `dss0` including a rhetorical event

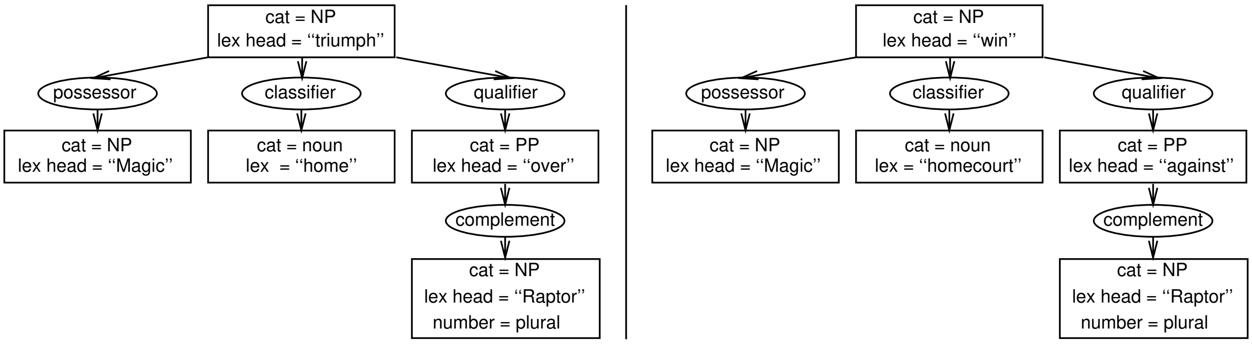

Figure 3.9: `dgs21` (left) and `dgs22` (right): two different lexicalizations of `sss2`

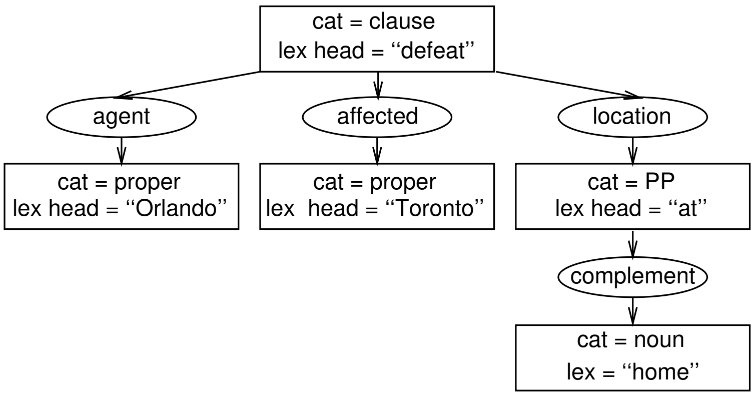

Figure 3.10: `dgs1`: an example lexicalization of `sss1`

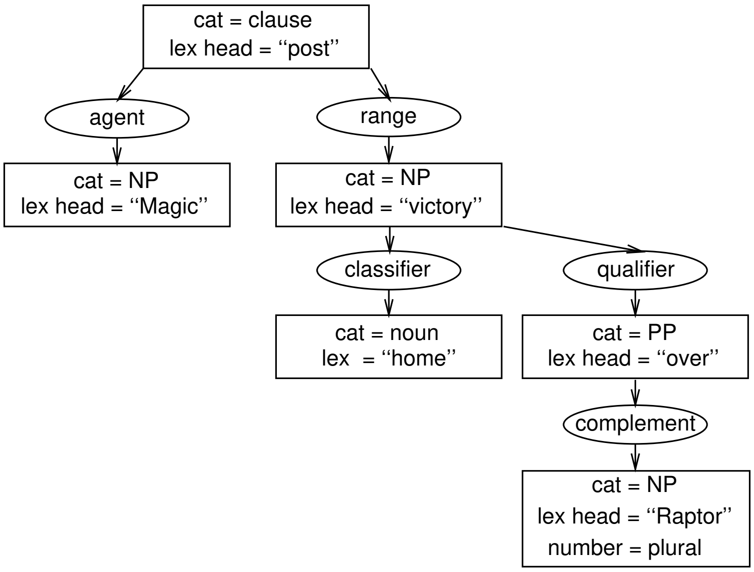

Figure 3.11: `dgs3`: an example lexicalization of `sss3`

In general, a DGS represents more than one phrase. For example, `dgs1` in Fig. 3.10 represents the lexicalization of `sss1` by the phrases:
"Orlando defeating Toronto at home" and
"At home, Orlando defeated Toronto" in addition to
"Orlando defeated Toronto at home".

For the sake of completeness, Fig. 3.11 gives the DGS lexicalizing `sss3` by the phrase:
"The Magic posted a home victory over the Raptors".

3.3.1.4 Draft representation

All three representation layers are needed in the draft representation. Architectural principles (3) and (4) defined in Section 3.2.2, concerned the draft representation. One required the presence of a purely conceptual layer independent of linguistic form. This is the DSS. The other required the presence of a surface layer reflecting realization choices. This is the DGS. Finally, as the intermediary layer that bridges the gap between these two end-layers, the SSS is also needed.

The draft representation is thus a triple <DSS,SSS,DGS>. Each of the last two elements in this triple is a complex structure with several constituents linked by various dependency relations. Since the mapping between any two of these layers is both one-to-many and non-isomorphic, the draft representation must also include explicit markers of the correspondence between the constituents of a given layer and the constituents of the next layer. Revising a syntactic structure without explicit knowledge of what semantic element each of its constituent realizes could have undesirable side-effects such as deletion of part of the original semantic message, introduction of ambiguities etc. An example of a three-layered draft representation is given in Fig. 3.12. The SSS arcs mark the correspondence between DGS and SSS constituents, while the DSS arcs mark the correspondence between SSS constituents and DSS elements. This particular three-layer structure represents phrases like: "The Magic's home triumph over the Raptors.

For the sake of simplicity, only the most important features of the draft representation are shown in this

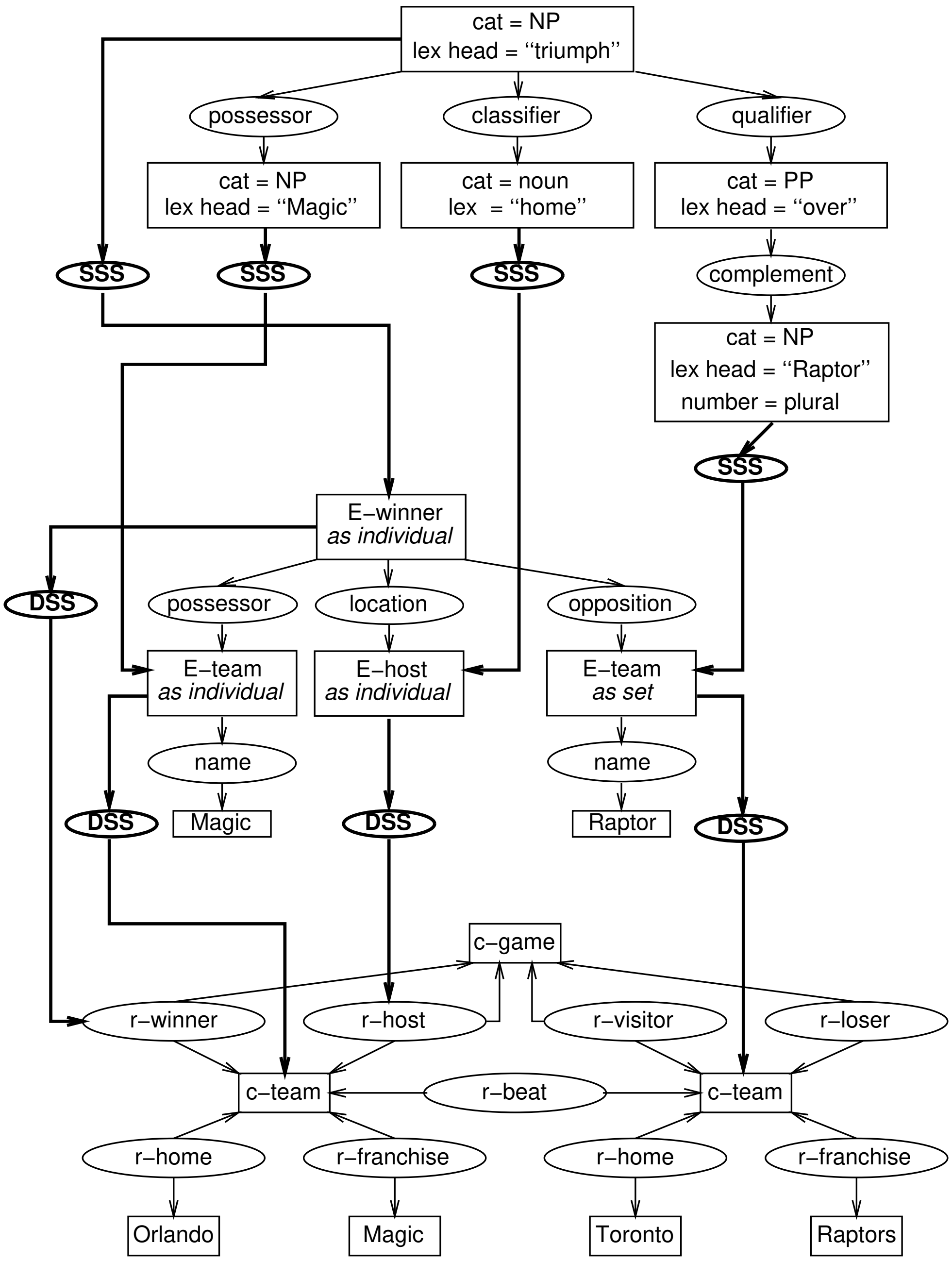

Figure 3.12: An example of three-layered draft representation

figure. In the practice of implementation, additional features are needed to exactly specify the phrase above as the current draft. These additional features are presented in Section 4.2 in the chapter describing the implementation of the STREAK system.

3.3.1.5 Advantages of the three-layer scheme

In a generation system, utterances must be represented by at least two layers: a semantic layer used for planning purposes and a syntactic layer used for syntactic purposes. In the representation scheme proposed here, there are two semantic layers: the DSS used for text planning purposes[8] and the SSS which bridges the gap between the DSS and the DGS. The need for a representation that bridges the gap between conceptual and syntactic processing has been advocated in detail by [Meteer 1990]. For alternative schemes using a single semantic representation, there are two possibilities: either the single semantic layer is like a DSS or it is like an SSS.

If it is like an SSS, the domain knowledge of the generator must be encoded in terms of linguistically motivated categories. This is the approach advocated by PENMAN's Upper-Model [Bateman *et al.* 1990], where the facts and entities of the domain knowledge base are viewed as instances of domain-independent categories defined in terms of linguistic ranks and thematic roles. The drawback of this approach is that it makes problematic the representation of floating facts at a variety of linguistic ranks and by a variety of thematic role structures.

Consider again the two paraphrases below:
(X) "*Orlando defeated Toronto at home*"
(Y) "*The Magic's home triumph over the Raptors*"

In the Upper-Model, the meaning of these two sentences would have to be represented as two distinct facts, instances of two different domain-independent categories: material action and possessed object. With such an approach, the only way to account for paraphrases like the two above is to artificially introduce in the domain knowledge base, fact duplicates corresponding to the different realization perspectives. In contrast, with the three layer scheme proposed here, the common meaning of such sentences is captured by a single DSS which is in then mapped onto different SSSs each representing a different realization perspective. X and Y thus share the same DSS, `dss` in Fig. 3.4, but have different SSSs: `sss1` in Fig. 3.5 for X and `sss2` in for Y (in the same figure). The key advantage of having two semantic representations is that each one assumes a single task: the DSS abstracts from linguistic form whereas the SSS abstracts from the domain. Paraphrases that cut across linguistic ranks like the two above, highlight the inherent conflicts between these two tasks which the Upper-Model approach attempts to reconcile within a single representation.

Note that the very idea of an Upper-Model is not questioned by this observation. Having a set of general concepts under which to attach the particular concepts of a given domain, is extremely helpful for developing a conceptual domain hierarchy. The problem arises with overusing linguistic criteria - and in particular linguistic rank and thematic roles - for defining these general concepts. The temptation for such overuse is great because there are not many non-linguistic criteria to fall back on for defining an Upper-Model (see [Lenat and Guha 1989] for an attempt to build a non-linguistically motivated general ontology). However, semantic linguistic categories are best viewed as constituting a "surface perspective" model rather than an "upper" model for the domain.

Without a DSS, a system would have difficulty handling paraphrases that cut across linguistic ranks. Systems with a single semantic representation that is like a DSS avoid this problem. However, such systems must perform a direct mapping from the DSS onto the DGS. This mapping is a complex task for the three following reasons:

- It involves several generation subtasks: phrase level content planning, lexicalization and semantic grammaticalization.
- It is one to many.
- It is non-isomorphic.

[8] Sentence planning occurs during the mapping from the DSS to the SSS.

The entire paraphrasing power of a sublanguage is captured by the mapping from a single DSS onto a whole set of synonymous DGSs. There are, however, three distinct aspects to that paraphrasing power:

- Syntagmatic paraphrasing, which involves choosing between synonymous word combinations that are associated with different syntactic categories and/or argument structures (*e.g., "Orlando defeated Toronto" vs. "Orlando's triumph over Toronto"*).
- Paradigmatic paraphrasing, which involves choosing between synonymous word combinations for a fixed syntactic category and argument structure (*e.g., "Orlando defeated Toronto" vs. "Orlando beat Toronto"*).
- Grammatical paraphrasing, which involves choosing between synonymous surface forms with the same open-class words but different syntactic properties (*e.g., "Orlando defeated Toronto" vs. "Orlando defeating Toronto"*).

In addition, the mapping from DSS to DGS is in general non-isomorphic as already mentioned in sections 3.3.1.2 and 3.3.1.3. This non-isomorphism is well illustrated in Fig. 3.12 containing all three representation layers for the same phrase. Non-isomorphism between semantic and syntactic structure has been noted in many domains (cf. [Talmy 1976], [Talmy 1983], [Zock 1988]) and has multiple aspects: several semantic elements can be conflated into a single syntactic element, some syntactic elements may be present for purely grammatical reasons without corresponding to any of the semantic elements, the syntactic element realizing the semantic head can be deeply embedded in the syntactic structure while the syntactic head may realize a semantic element embedded in the semantic structure, etc.

The main advantage of having an SSS layer, is that it breaks down the overall complexity of mapping a DSS onto a DGS into two stages, each involving a smaller, more manageable number of decisions. In particular:

- Different generation subtasks are handled at different stages: phrase level planning is handled during DSS$\mapsto$SSS mapping while lexicalization and semantic grammaticalization are handled during SSS$\mapsto$DGS mapping.
- Different aspects of paraphrasing are captured by different stages: syntagmatic paraphrasing is captured by the DSS$\mapsto$SSS mapping while paradigmatic and grammatical paraphrasing are captured by the SSS$\mapsto$DGS mapping.
- Different sources of non-isomorphism are handled at different stages.

Another use for a double semantic representation is to distinguish between implicit and explicit realization of content as already discussed in Section 3.4. [Dale 1992] adopted a double semantic representation in EPICURE specifically for such purpose.

3.3.2 Processing components and overall control

As shown in Fig. 3.13 there are six components in the new generation architecture I propose: the fact generator, the discourse planner, the phrase planner, the lexicalizer, the reviser and the syntactic grammar. Generation proceeds in two passes. During the first pass an initial draft is built containing only the fixed facts. During the second pass, floating facts (including historical facts) are incrementally incorporated, by fitting them opportunistically within the current draft. The processing components and internal representations involved in the draft construction are shown in figure 3.14. Those involved in the draft revision and the final natural language output are shown in figure 3.15[9].

The first pass starts upon reception of a set of new statistics to report in tabular form. This table of numbers is read by the fact generator which creates a record for each of them[10]. Depending on the

[9] Note how several processing components and internal representations play a role in both passes.

[10] Or part of them in a domain where the irrelevance of some facts can be established without considering the domain's historical background.

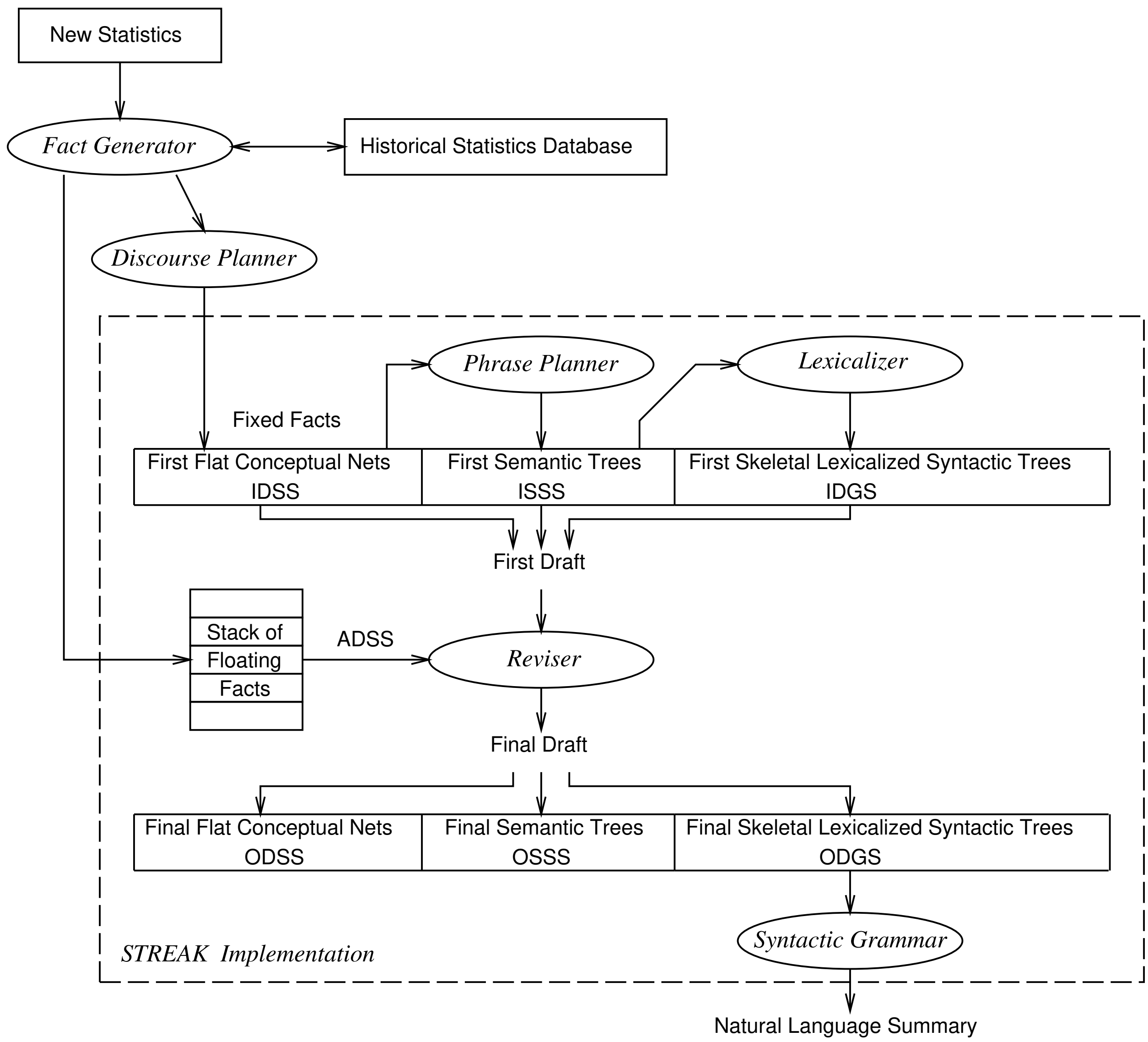

Figure 3.13: A revision-based generation architecture: overall picture

implementation, this record could be a LISP or C structure or, as in the case of ANA, an OPS5 working memory element. The fact generator then queries a database of historical statistics to retrieve for each new statistic a set of related historical statistics. In the sports domain, access to such historical databases through a modem is offered by several companies that specialize in such services. This historical data is used to assess the relevance of the new data and vice-versa. This allows the fact generator to select a set of candidate facts, both new and historical, to potentially include in the natural language summary. It then takes the record representing each fact and creates a DSS for each of them. This second reformatting task involves deducing the redundant information needed in the DSS for the generator to be able to perform cross-ranking paraphrasing. The need for such additional information for language generation (a pervasive problem in generation, cf. [McKeown and Swartout 1987]) was explained in the previous section. For example in `dss1` in Fig. 3.6 of the previous section, the relations `r-loser` and `r-beat` can be deduced from records for the relations `r-winner`, `r-host` and `r-visitor`. The fact generator outputs a list of a set of facts.

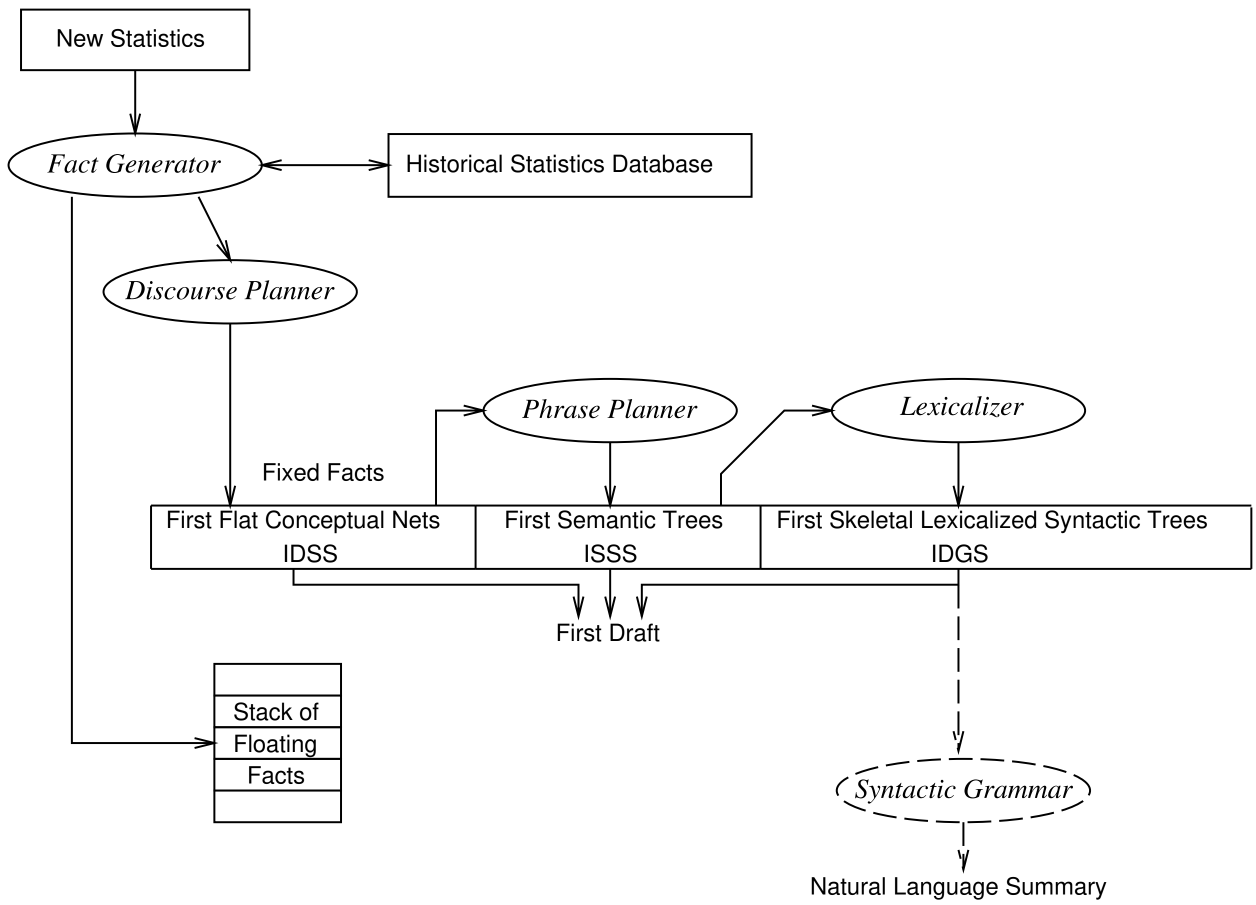

Figure 3.14: A revision-based generation architecture: **draft time**

Each set of facts contains a new fact and possibly some related historical facts. Each fact is represented by a DSS augmented by a relevance grade. The fact generator is the only component in this architecture that is particular to quantitative data summarization. For other classes of applications, another type of underlying application program interface would be needed to produce the corresponding list of set of DSSs. The fact generator is out of the scope of this thesis and has therefore not been implemented in the prototype system STREAK. For such an implementation, it would probably be best to further decompose it into several sub-components: one which reads the input table and produces fact records, one which interfaces with the historical database, one which gives a relevance grade to each new or historical fact and one which produces a DSS for each preselected fact.

The output of the fact generator is passed to the discourse planner. The discourse planner traverses a textual schema [McKeown 1985] encoding, for each sentential slot in the report, which type of fixed facts it must contain. Recall from Section 2.4 that fixed facts are those present in every report and in the same sentence over the human-written model report corpus. They are contrasted with floating facts which are present only in some reports and in different sentences over of the human-written model report corpus. In the basketball domain, all historical facts are floating. The most important new facts are fixed but the secondary ones are also floating. For each sentential slot, the discourse planner searches for the fixed facts that the schema indicates for that slot in the list of set of facts it received in input . For example in the basketball domain, such a schema would instruct the discourse planner to look for four fixed facts for the lead sentence slot: the individual statistic with the higher relevance grade, the result of the game, its location

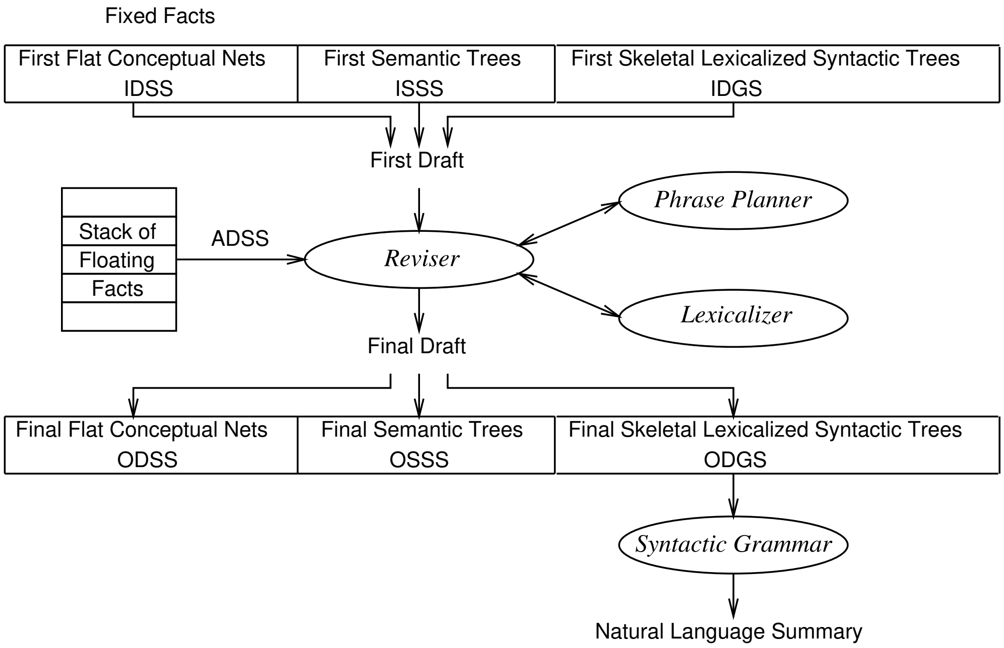

Figure 3.15: A revision-based generation architecture: **revision time**

and its date. The desired number of sentences of the summary to generate is given as an input parameter constraining the textual schema traversal. When the discourse planner reaches the end of the instantiated schema, the facts in its input list that did not fit in any fixed slot are put into a stack of floating facts and ordered by decreasing relevance grade. The output of discourse planner is thus twofold:

- A list of Initial DSSs (IDSS), representing the fixed facts for each sentential slot. These facts must be included in the first draft.
- A stack of Additional DSSs (ADSS) representing the floating facts for the whole report. These facts are to be incorporated opportunistically in the final draft and are passed to the reviser.

Only the first part of the discourse planner output is used at draft time. For each flat conceptual network in this IDSS the phrase planner is called and maps this network onto a corresponding semantic tree which represents high-level sentence structure. In the process it chooses:

- Which element of the conceptual network will be explicitly realized by a linguistic element in the corresponding sentence and which to leave implicit.
- The dependency relations among the elements chosen for explicit realization.

The result is a list of Initial SSSs (ISSS). The lexicalizer is then called on each semantic tree in this ISSS and maps it onto a corresponding lexicalized syntactic tree. In the process it chooses for each linguistic constituent:

- Its open-class words.
- Its syntactic category.
- The syntactic features that differ from the set of defaults provided by the syntactic grammar for that category.

The result is a list of Initial DGSs (IDGS). An input switch parameter can be used to set the generator to an incremental mode in which each intermediate draft is displayed. If this switch is on, the syntactic grammar is called on each element in IDGS and an initial draft gets generated. Each sentence in this initial textual draft is internally represented by a <DSS,SSS,DGS> triplet. The list of these triplets, together with the ADSS stack, constitutes the output of the drafting pass and the input to the reviser.

For each draft sentence, the reviser pops an ADSS from the stack of floating facts and attempts to incorporate it to the sentence using the revision operations presented in Section 2.5. These operations are generic and only specify where (*i.e.,* on which subconstituent) and how (*i.e.,* using what type of syntactic attachment) to add the new constituent realizing the ADSS inside the draft sentence For determining the internal content organization, wording and syntactic form of this new constituent the reviser does not rely on revision rules. It instead calls the phrase planner and the lexicalizer to carry out those tasks in the context of the revision, as indicated by the arrows linking the reviser to the phrase planner and the lexical chooser in Fig. 3.15.

Domain specific stylistic and space constraints monitor the revised draft. After each revision increment, the DGS of the revised draft is examined to determine whether it has reached the maximum complexity observed for this sentence slot in the corpus of human-written model summaries. For example in the basketball domain, such constraints indicate that the lead sentence should not exceed 46 words in length and 10 levels of constituent embedding in syntactic depth. When such thresholds are reached, the reviser proceeds to the next draft sentence. The ADSSs that were popped from the stack during the revision of the previous sentence but which could not fit there due to either lack of a matching revision rule or lack of space are then pushed back onto the floating fact stack. When the generator is called in incremental mode, the syntactic grammar is called on each revised DGS and the entire revised draft is displayed anew. Otherwise, the syntactic grammar is called only once for each report sentence after the entire revision process has been completed and the intermediate drafts are not displayed.

3.4 Generation subtasks revisited

In this subsection I explain how the generation subtasks defined in Section 3.1 are distributed among the various processing components of the architecture proposed in Section 3.3 and briefly discuss the general applicability of the new architecture.

In the new language architecture presented in the previous section:

- *Content production* is carried out by the fact generator.
- *Content selection* is carried out in two rounds. The first round is performed under encyclopedic domain constraints by the fact generator. The second round is performed under syntactic, lexical and space constraints by the reviser.
- *Discourse level content organization* is carried out by the discourse planner.
- *Phrase level content organization* is carried out by the phrase planner. It thus occurs in two rounds: a first round when the phrase planner is called on the initial IDSS and a second round when it is called from the reviser.
- *Lexicalization* is carried out by the lexicalizer and similarly occurs in two rounds, one at draft-time and one at revision-time.
- *Semantic grammaticalization* is carried out in part by the phrase planner and in part by the lexicalizer. It this also occurs in two rounds, one at draft-time and one at revision-time.
- *Morpho-syntactic grammaticalization and linearization* are both carried out by the syntactic grammar.

The new generation architecture presented in the previous section was primarily motivated with one class of language generation application in mind: summarizing quantitative data. However, only one component in this architecture is specific to this class of applications: the fact generator. For other classes of applications, another type of underlying application program interface would be needed for content production, (*i.e.,* for generating the DSSs input to the rest of the system). The rest of the architecture is general and could be used for any text generation system. It allows for more flexibility and compositionality in the generation process but it is also more complex than most previously proposed architectures. What are the general circumstances in which these improvements to the generation process are really needed? I believe, any one of the following circumstances demands in itself either the added flexibility or the added compositionality provided by this new architecture:

- The application domain sublanguage contains both very complex sentences and a large number of paraphrasing forms for them.
- Stylistic surface form constraints (*e.g.,* space limitation) need to be tightly monitored.
- A large proportion of the application domain concepts are floating[11] and of fairly even relevance.
- All the concepts from at least one important content class are of floating nature.

3.5 Summary

In this section I have presented a new system architecture with the ability to generate very complex sentences compositionally and to convey historical information in the flexible and concise way it appears in human-written summaries. With this architecture, generation proceeds in two passes. The first pass builds an initial report draft organizing and realizing fixed semantic elements. The second pass incrementally revises this draft to opportunistically incorporate floating semantic elements, in particular historical facts. Internally, the draft is represented at three levels of abstraction. The most abstract level is purely conceptual. At the next level domain concepts are mapped to general semantic linguistic categories. These abstract linguistic resources are then mapped to specific lexical items and syntactic forms at the next level. The natural language report is then produced by passing this last level to a portable syntactic grammar.

[11] *i.e.,* are mentioned in only a subset of the model text corpus and in different sentential slots in across this corpus.

Chapter 4

Implementation prototype: the STREAK generator

4.1 Goals and scope of the implementation

In this chapter, I present the implementation of the language generator STREAK (Surface Text Reviser Expressing Additional Knowledge). This implementation has two main goals: demonstrating the operationality of the revision tools extracted from the corpus analysis presented in Chapter 2 and demonstrating the practicality of the new generation system architecture proposed in Chapter 3. STREAK is a research prototype intended as a testbed for the new draft and revision approach to language generation advocated in this thesis, not a finished product intended for everyday use in a real-world application. In order to keep the implementation effort to a manageable size, STREAK focuses on:

- The core of the domain sublanguage, *i.e.,* the lead sentences that were the object of the systematic in-depth analysis presented in Chapter 2.
- The most original components of the complete text generation architecture proposed in Section 3, *i.e.,* those involved in the revision pass of the overall generation process.

Recall from figures 3.14 and 3.15 that two components in that architecture are used in both the initial draft pass and the subsequent revision pass: the phrase planner and the lexicalizer. They are called both at draft time to realize the fixed facts and again at revision time to realize the floating facts. In addition to the complete revision pass, the surface realization part of the draft pass is thus also implemented in STREAK.

Input to STREAK is hand-coded and constitutes of:

- A initial DSS (IDSS) representing all the fixed facts of the lead sentence to generate.
- A list of ADSSs by order of decreasing importance, with each element in the list representing a floating facts to attempt to opportunistically incorporate to the lead sentence.

STREAK generates either one or a set of synonymous complex sentences concisely expressing all the facts in the IDSS plus as many facts from the ADSS list that could be fit in without exceeding the maximum word length or syntactic depth observed in the corpus lead sentences.

STREAK is implemented using the FUF/SURGE package [Elhadad 1993a], [Elhadad 1993b] for developing language generation application. This package is presented in detail in Appendix B. It consists of:

- FUF [Elhadad 1993b] [Elhadad 1993a], a special-purpose programming language for text generation based on functional unification.
- SURGE, a wide-coverage grammar of English implemented in FUF and usable as a portable front-end for syntactic processing.

FUF is the *formalism* part of the package, a language in which to encode the various knowledge sources needed by a generator. SURGE is the *data* part of the package, one already encoded knowledge source usable by any generator. Using the FUF/SURGE package, implementing a generation system thus consists of decomposing *non*-syntactic processing into sub-processes and encoding in FUF the knowledge sources for each of these sub-processes. In the case of STREAK, *non-syntactic* processing is decomposed in phrase planning, lexicalization and revision. STREAK thus relies on three *non*-syntactic knowledge sources: the phrase planning rule base, the lexicalization rule base and the revision rule base. These three knowledge sources are encoded in FUF.

Both FUF and SURGE had to be extended during the development of STREAK to meet some of its special needs. First, the non-monotonicity of STREAK required the implementation of new FUF operators to cut and paste functional descriptions aside from unifying them. These extensions to FUF were implemented by Elhadad. Second, while SURGE already covered a wide variety of syntactic forms for simple clauses[1] it only covered a very few for complex sentences, which aggregate several such clauses. It also did not cover the specialized nominals of quantitative domains. I implemented sizeable extensions to SURGE to attain wide coverage for complex sentences and quantitative nominals. This set of extensions goes far beyond what was needed for the sole needs of STREAK and constitutes in itself a significant, though not central, contribution of this thesis. It was not simply an implementation matter but required to cross-examine the descriptive work of several non-computational linguists and integrate their respective analysis within the unifying computational framework of SURGE. For quantitative nominals, some constructs I observed in newswire report corpora were not mentioned in the linguistic literature and I had to come up with my own analysis. The version of SURGE resulting from these extensions has since been used for other generation applications in addition to STREAK: automated documentation for the activity of telephone network planning engineers [Kukich *et al.* 1994], verbal descriptions of visual scenes [Abella 1994] and generation of business letters.

The two sets of extensions to the FUF/SURGE package are described in detail Appendix B. They were essentially preparatory work paving the way for development of STREAK as a prototype for a particular application. The present chapter describes this development itself. To that purpose, I first come back to each layer of internal draft representation abstractly defined in Section 3.3 and show how it is encoded in detail as a FUF description[2]. I then present in turn the FUF implementation of each processing component proper to the STREAK generator (*i.e.,* the phrase planner, the lexicalizer and the reviser). Finally, I comment in detail on the trace of two runs of STREAK on selected examples (more selected examples runs are discussed in Appendix C).

4.2 The internal representation languages

In STREAK, a natural language utterance is specified at three different layers of representations:

- The Deep Semantic Specification (DSS) which is a flat conceptual network.
- The Surface Semantic Specification (SSS) which is a structured semantic tree.
- The Deep Grammatical Specification (DGS) which is a skeletal lexico-syntactic tree.

These three layers were described at the abstract design level in Section 3.3.1. The DGS is described at the detailed implementation level during the presentation of SURGE in sections B.2 and B.3 of Appendix B. The present section describes the DSS and SSS at the detailed implementation level.

[1] In fact as wide as any other available syntactic processing front-ends for language generation.

[2] Since FUF is a declarative language with little syntax, the examples of the next section should be understandable in an intuitive way. However, for reader interested in grasping them in the details of the implementation, a self-contained introduction to FUF can be found in Section B.1 of Appendix B. It contains all the FUF syntax and special features appearing in the examples of this thesis.

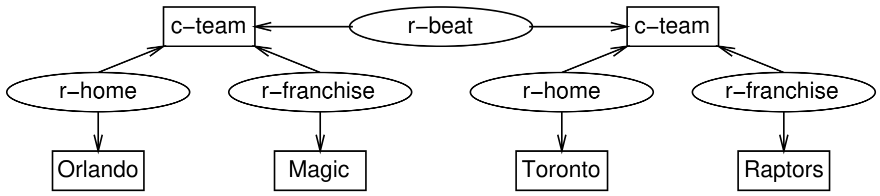

Figure 4.1: A very simple conceptual network

4.2.1 The DSS language

As explained in Section 3.3.1, at the abstract design level, the DSS is a flat conceptual network. However, at the implementation level, STREAK relies on a single data structure: the Functional Description (FD) (presented in detail in Appendix B) for encoding each of the three abstract representation layers. In the following sections, I first describe the general issue of encoding a flat conceptual network as an FD. I then describe the specific set of features used to encode the subnets representing the fixed and then floating (including historical) facts in STREAK's particular application domain.

4.2.1.1 Representing a flat conceptual network as an FD

Each element in a conceptual subnet is represented as an FD with the following attributes:

- `deepsemcat` (DEEP SEMantic CATegory) whose value is `entity` for a concept node and `relation` for a role arc.
- `concept` for concept nodes and `role` for role arcs.
- `token` identifying a particular instance of a given concept or role.
- `attrs` (ATTRibuteS) for a concept node, specifying the atomic roles (*i.e.*, those that are not recursively filled by a concept) of the concept.
- `args` (ARGumentS) for a role arc, pointing to the atomic value(s) or concept(s) that the arc relates.

A conceptual subnet is itself represented as an FD with two attributes:

- `ents` (ENTitieS) containing the FD representation of each concept node in the subnet.
- `rels` (RELationS) containing the FD representation of each role arc in the subnet.

For example the subnet of Fig. 4.1, where concepts are in square boxes and roles in oval boxes, translates into the FD at the top of Fig. 4.2. The subnet and corresponding FD are dual representations of the content of paraphrases such as:

1. "The Orlando Magic defeated the Toronto Raptors",
2. "The Orlando Magic, who triumphed over the Toronto Raptors",
3. "The Toronto Raptors, who were defeated by the Orlando Magic".

Note the difference in the subnet between, on the one hand, full-fledged concepts, such as `c-team`, which are prefixed by "`c-`" and linked to several roles, and on the other hand, atomic values such as `Orlando` that are linked to only a single role. This difference is paralleled among roles. Full-fledged roles like `r-beat` relate two full-fledged concepts, while atomic roles like `r-home` relate a full-fledged concept with an atomic value. As shown in FD1, these differences are reflected in the FD representation of a network:

```
FD1: FD representation of the flat conceptual subnet of Fig. 4.1 p.62
((ents ((winner ((deepsemcat entity)
                 (concept team)
                 (token magic-vs-raptors-winner)
                 (attrs ((home "Orlando") (franchise "Magic")))))
        (loser ((deepsemcat entity)
                (concept team)
                (token magic-vs-raptors-loser)
                (attrs ((home "Toronto") (franchise "Raptors")))))))
 (rels ((result ((deepsemcat relation)
                 (concept beat)
                 (token magic-vs-raptors-result)
                 (args ((winner {^4 ents winner})
                        (loser {^4 ents loser}))))))))

FD2: Structured (i.e., unflat) FD representation of the same content.
((deepsemcat relation)
 (concept beat)
 (token magic-vs-raptors-result)
 (args ((winner ((deepsemcat entity)
                 (concept team)
                 (token magic-vs-raptors-winner)
                 (attrs ((home "Orlando") (franchise "Magic")))))
        (loser ((deepsemcat entity)
                (concept team)
                (token magic-vs-raptors-loser)
                (attrs ((home "Toronto") (franchise "Raptors"))))))))
```

Figure 4.2: Representing a flat conceptual net as an FD

atomic values are LISP atoms or strings, whereas full-fledged concepts are structured FDs appearing under the `ents` attribute. Similarly, atomic roles are sub-attributes appearing under the `attrs` attribute inside FDs representing full-fledged concepts and they are filled with atomic values, whereas full-fledged roles are structured FDs appearing under the `rels` attribute, with `args` sub-attributes pointing to full-fledged concepts.

The bottom of Fig. 4.2 contains FD2, an alternative FD representation for the content of FD1 at the top of the same figure. In this alternative representation, the top-level features `ents` and `rels` have been suppressed and the path values of the `args` sub-attributes replaced by embedded FDs. Although more concise than FD1, this representation no longer represents a *flat* network, but instead a structured *tree* with the particular bias of giving special head status to the `c-beat` element of the network. From such tree structure only the first of the three paraphrases above that convey the content of the flat network could be generated, since in the two others, it is one of the two `c-team` elements of the network that heads the linguistic structure.

The idea of representing flat networks as FDs by listing concepts and roles under different top-level features and indicating the link between them through paths instead of embedding is due to [Elhadad 1993b].

```
FD3: STREAK input representation for the fact represented in isolation by FD1.
((deepsemcat entity)
 (concept game)
 (token magic-vs-raptor)
 (attrs ((results ((ents ((winner ((deepsemcat entity)
                                   (concept team)
                                   (token magic-vs-raptors-winner)
                                   (attrs ((home "Orlando") (franchise "Magic")))))
                         (loser ((deepsemcat entity)
                                  (concept team)
                                  (token magic-vs-raptors-loser)
                                  (attrs ((home "Toronto") (franchise "Raptors")))))))
                   (rels ((result ((deepsemcat relation)
                                   (concept beat)
                                   (token magic-vs-raptors-result)
                                   (args ((winner {^4 ents winner})
                                          (loser {^4 ents loser}))))))))))))
```

Figure 4.3: Example of fact in input format in STREAK

4.2.1.2 Representing fixed facts in the STREAK domain

Having described the domain-independent scheme for representing a DSS network as an FD, I now turn to the specific set of features used for describing the sports domain of STREAK. In this domain, each report generated describes a given game. Every fact the report contains thus directly or indirectly concerns that game. Direct facts come from a standard box-score summing up the statistics of the game and indirect facts come from a database about the corresponding sports league and include historical statistics compiled over time and across multiple games.

STREAK takes advantage of both this common theme in each report and of the ability of FUF to work with partial information to represent any content unit as a partial description of the reported game. Consider for example the subnet of Fig. 4.1. It provides the result of the reported game. *FD1* at the top of Fig. 4.2 represents this fact in isolation. The actual input to STREAK for this fact is shown in Fig. 4.3. It views this fact not in isolation, but instead as one piece of information about the game to report. This view allows the incremental addition of content units, which characterizes the revision-based generation model defended in this thesis, to reduce, at the DSS layer, to a succession of simple unification operations.

The skeletal frame of a game description is derived from a series of observations I made about the organization of content inside the corpus of lead sentences that served as target output for STREAK. These observations, first presented in Section 2.4, are the following:

- They are two different types of facts: *fixed* facts, that are obligatory in any lead sentence, and *floating* facts, that only occasionally appear in the lead sentence.
- There are four fixed facts: the main statistic of a player from the winning team, the game result (including the final score-line), the location of the game and its date.
- All corpus lead sentences, whether the basic ones containing only these four fixed facts, or the more complex ones that contain floating facts as well, consist of two main syntactic constituents: one containing the main statistic and one containing the game result.
- The location is always conveyed separately as a header, while the date indifferently appears embedded in either of these two main constituents.

- The floating facts cluster around either the main statistic or the game result, in a way pre-determined by their semantic class (*e.g.,* additional statistics of a player of the winning team systematically cluster around the main statistic, while result streaks systematically clustered around the game result).

Let us now see how these observations are reflected in the skeletal frame of a game description. The input FD for a basic draft sentence containing only fixed facts is shown in Fig. 4.4. The game location (abbreviated `addr` for address) and its date, that together define the general circumstances of the game and do not influence the sentence structure, are grouped under the attribute `setting`. The rest of the game description consists of two subnets, one for the main statistic cluster under the attribute `stats` and one for the game result cluster under the attribute `results`.

Compare FD4 of Fig. 4.4 with:

- FD1, where each full-fledged role is explicitly represented by an FD under the `rels` attribute which `args` sub-attributes are filled with paths pointing to the concepts the role relates,
- FD2, where each such role is represented by an attribute whose sub-attributes are directly filled with the FDs describing the concepts to which the role relates,

reveals that the FD representation of a draft input in STREAK, is really an hybrid of the pure pointer approach of FD1 and the pure embedding approach of FD2.

The embedding approach is more concise but prevents the generation of phrases where a role that is not explicitly represented as an FD is explicitly conveyed as a linguistic constituent. In the draft input of Fig. 4.4, the `address, date, score, host` and `visitor` *roles* are not explicitly represented because the target sublanguage of STREAK observed in the corpus did *not* contain phrases explicitly mentioning these roles such as:

- "the game opposing the Magic to the Raptors **was held** in Orlando",
- "**the date** of the game was Saturday",
- "**the score** of the game was 101-89",
- "Orlando **was hosting** this game against the Raptors"

Instead, only the `address, date, score, host` and `visitor` *concepts* were explicitly mentioned, with the role linking them to the game implicitly expressed by the sentence structure as in:
"**Orlando, FL** – Shaquille O'Neal scored 37 points **Friday night**, lifting the **Orlando** Magic to a **101 - 89** victory over the Toronto Raptors."

The approach for the input content representation in STREAK is thus to represent explicitly as FDs only those roles that are explicitly realized by a linguistic constituent in at least one phrase of the target sublanguage. In the sublanguage of the basic lead sentences summarizing basketball games, this is the case for the *winner, loser* and *result* relations as demonstrated by the following example paraphrases:

- "Orlando's **victory** over Toronto"
- "Toronto's **loss** at the hand of Orlando"
- "Orlando **triumphed over** Toronto"

For a basic main statistic cluster, the role `stat0` representing this statistic is the only one needed. It is in general explicitly realized as in: "O'Neal **scored** 37 points"
but can also be left implicit as in: "O'Neal's 37 points".

4.2.1.3 Representing floating and historical facts in the STREAK domain

The target sublanguage of STREAK contains five different types of floating facts:

```
``Orlando, FL -- Shaquille O'Neal scored 37 points Friday night,
lifting the Orlando Magic to a 101 - 89 victory over the Toronto Raptors.''

FD4:
((deepsemcat entity)
 (concept game) (token tor-at-orl)
 (attrs ((setting ((ents (;; ``Orlando, FL''
                          (addr ((deepsemcat entity)
                                 (concept address) (token orl-fl)
                                 (attrs ((city "Orlando") (state "FL")))))

                          ;; ``Friday night''
                          (date ((deepsemcat entity)
                                 (concept date) (token fri-nite)
                                 (attrs ((day-name "Friday") (day-part night)))))))))

         ;; ``Shaquille O'Neal scored 37 points''
         (stats ((ents ((stat0-ca ((deepsemcat entity)
                                   (concept player) (token oneal)
                                   (attrs ((first-name "Shaquille") (last-name "O'Neal")))))
                        (stat0 ((deepsemcat entity)
                                (concept game-stat) (token oneal-pt-vs-den)
                                (attrs ((value 37) (unit pt)))))))
                 (rels ((stat0 ((deepsemcat relation)
                                (role game-stat-rel) (token oneal-scoring-vs-den)
                                (args ((carrier {^4 ents stat0-ca}) (stat {^4 ents stat0})))))))))

         ;; ``lifting the Orlando Magic to a 101 - 89 victory over the Toronto Raptors.''
         (results ((ents ((host ((deepsemcat entity)
                                 (concept team) (token magic)
                                 (attrs ((home "Orlando") (franchise "Magic")))))
                          (visitor ((deepsemcat entity)
                                    (concept team) (token raptors)
                                    (attrs ((home "Toronto") (franchise "Raptor")))))
                          (score ((deepsemcat entity)
                                  (concept score) (token tor-at-orl-score)
                                  (attrs ((win 101) (lose 89)))))))
                   (rels ((winner ((deepsemcat relation)
                                   (role winner) (token tor-at-orl-winner)
                                   (args ((game top-level) (winner {^4 ents host})))))
                          (loser ((deepsemcat relation)
                                  (role loser) (token tor-at-orl-loser)
                                  (args ((game top-level) (loser {^4 ents visitor})))))
                          (result ((deepsemcat relation)
                                   (role beat) (token tor-at-orl-result)
                                   (args ((winner {^4 ents host}) (loser {^4 ents visitor})))))))))))))
```

Figure 4.4: Example of input FD for basic draft

- Additional game statistics (first order, non-historical fact)
- Non-statistic background properties (first order, non-historical fact)
- Game statistic records (second order, historical fact)
- Game result streaks (second order, historical fact)
- Streak records (third order, historical fact)

I review the FD representation of each of these types in turn in the following subsections.

4.2.1.3.1 Additional statistics Each additional statistic is distinguished by an integer inversely proportional to their relevance. The subnet representing any statistic, either the main statistic or an additional one, consists of a single `statN` relation with two arguments, one pointing to the `statN` concept representing the value of the statistic and the other pointing to the `statN-ca` (STATistic number N's CArrier) concepts representing its carrier (*i.e.,* the player or team whose performance is summed-up by the statistic). The FD under `stats` in Fig. 4.4 is an example of such subnet.

4.2.1.3.2 Background properties Similar in structure to game statistics, background properties also consists of a single role whose arguments points to two concepts, one describing the property and the other its carrier. For example, the reserve status of the player Jay Humphries is represented by the following FD:

```
((deepsemcat entity)
 (concept game)
 (token uta-at-bos)
 (attrs ((stats ((ents ((stat1-ca-status ((deepsemcat entity)
                                          (concept reserve)
                                          (token humphries-reserve)))
                        (stat1-ca ((deepsemcat entity)
                                   (concept player)
                                   (token humphries)
                                   (attrs ((first-name "Jay") (last-name "Humphries"))))))
                 (rels ((stat1-ca-status ((deepsemcat relation)
                                          (role player-status)
                                          (token humphries-status)
                                          (args ((player {^4 ents stat1-ca})
                                                 (status {^4 ents stat1-ca-status}))))))))
```

Note how the property carrier is identified by way of the statistic that he performed. This reflects the fact that in the target sublanguage, such background fact is always opportunistically woven to the expression of a statistic as in:
"Jay Humphries **came off the bench** to score 24 points" or,
"Jay Humphries scored 24 points **off the bench**" or,
"**Reserve** Jay Humpries scored 24 points",
instead of a separate sentence such as: "Jay Humphries is a reserve player".

4.2.1.3.3 Statistic records A record update relates a new statistic to an *historical statistic*. An historical statistic is an abstract *set* of statistics. As opposed to an isolated statistic, which consists of a single relation, an historical statistic is a complex conceptual subnet relating multiple concepts:

- `duration`: the time span over which the set of statistics has been compiled.
- `gen-elt`: the GENeric ELemenT of the set, an abstract specification of the particular type of statistics that the set contains.
- `gen-elt-ca`: the CArrier of the GENeric ELemenT, common to all statistics in the set.

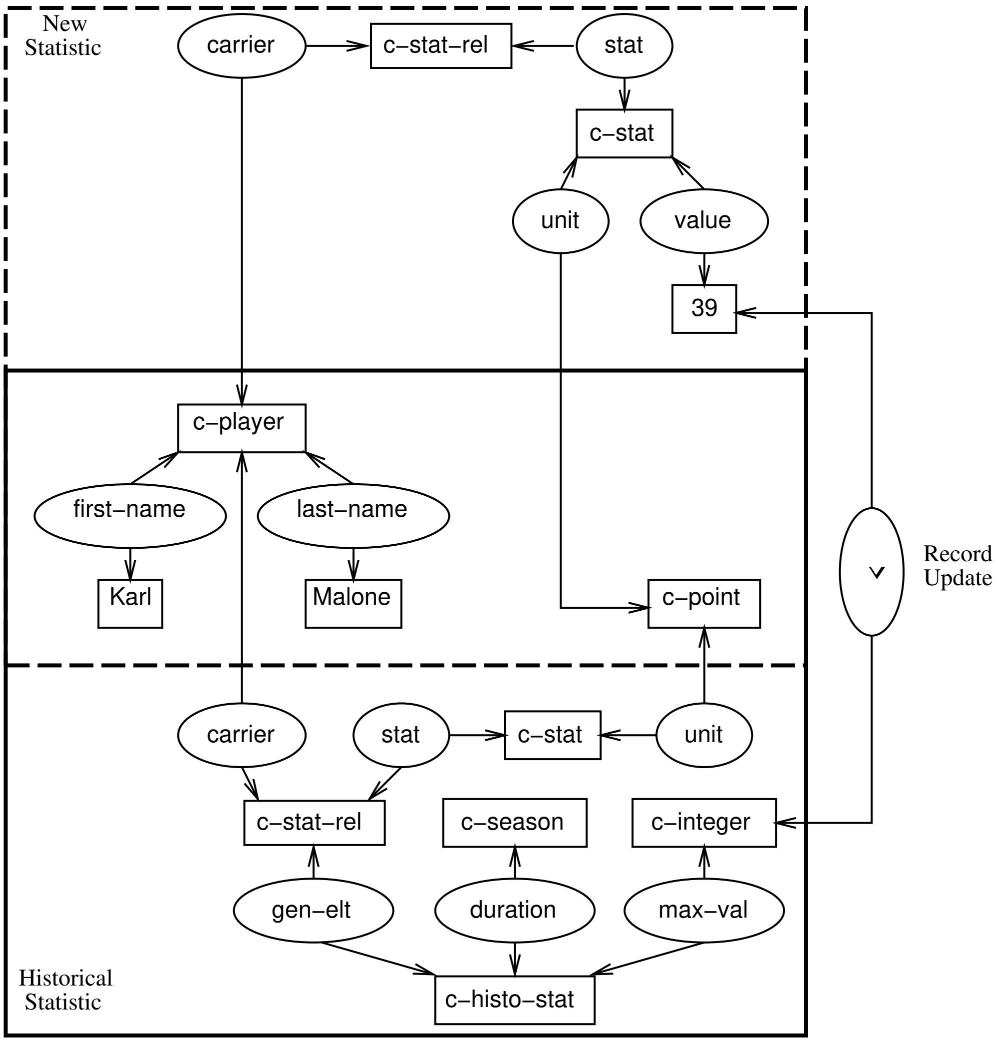

Figure 4.5: Example of conceptual network containing an historical record fact

- `max-val` (or `min-val`): the MAXimum (or MINimum) value for the particular of type statistic specified by `gen-elt` and `gen-elt-ca` and over the period of time specified by `duration`.

This internal complexity of an historical statistic makes a statistic record update a *second* order fact, as opposed to first order facts that relate only single nodes in the conceptual net such as the additional statistics or background properties presented above. An example subnet representing an historical statistic in is given in Fig. 4.5 inside the bottom square box. It abstractly represents the set of Karl Malone's scoring statistics over the current season.

The top subnet in the dotted box of the figure contains the new statistic onto which STREAK attaches the related historical one. This attachment is rendered possible by the fact that both facts share some elements, in the case at hand, the carrier of the statistic and its unit (in the intersecting portion of both boxes). The record update is shown by the arc containing the ">" sign which express that the new statistic value surpasses the maximum value of the historical statistic. The FD encoding of this statistic record update subnet is given in Fig. 4.6.

```
;; FD representing the record update fact conveyed by the uppercased constituent in the paraphrases:
;; ``Karl Malone scored A SEASON HIGH 28 points''
;; ``Karl Malone's SEASON BEST 28 points''
;; ``Karl Malone BROKE HIS SEASON RECORD WITH 28 points''

((deepsemcat entity)
 (concept game)
 (token uta-at-lac)
 (attrs ((stats ((ents ((histo-stat0 ((deepsemcat entity)
                                      (concept histo-stat)
                                      (token kmalone-pt-at-lac-ref-set)))
                        (histo-stat0-duration ((deepsemcat entity)
                                               (concept season)
                                               (token kmalone-pt-at-lac-ref-set-duration)))
                        (histo-stat0-gen-elt ((deepsemcat entity)
                                              (concept game-stat)
                                              (attrs ((unit pt)))))
                        (histo-stat0-gen-elt-ca ((deepsemcat entity)
                                                 (concept player)
                                                 (token kmalone)
                                                 (attrs ((first-name "Karl") (last-name "Malone")))))
                        (histo-stat0-extr ((deepsemcat entity)
                                           (concept integer)
                                           (token kmalone-pt-at-lac-ref-set-extr)))))
                 (rels ((histo-stat0-duration  ((deepsemcat relation)
                                                (role duration)
                                                (token kmalone-pt-at-la-ref-set-duration-rel)
                                                (args ((set {^4 ents histo-stat0})
                                                       (duration {^4 histo-stat0-duration})))))
                        (histo-stat0-gen-elt ((deepsemcat relation)
                                              (role gen-elt)
                                              (token kmalone-pt-at-lac-ref-set-gen-elt)
                                              (args ((set {^4 ents histo-stat0})
                                                     (gen-elt {^4 ents histo-stat0-gen-elt})))))
                        (histo-stat0-gen-elt-ca ((deepsemcat relation)
                                                 (role game-stat-rel)
                                                 (token kmalone-pt-at-lac-ref-set-gen-elt-ca)
                                                 (args ((carrier {^4 ents histo-stat0-gen-elt-ca})
                                                        (stat {^4 ents histo-stat0-gen-elt})))))
                        (histo-stat0-extr ((deepsemcat relation)
                                           (role max-val)
                                           (token kmalone-pt-at-lac-ref-set-extr-rel)
                                           (args ((extr-val {^4 ents histo-stat0-extr})
                                                  (set {^4 ents histo-stat0})))))
                        (histo-stat0-update
                         ((deepsemcat relation)
                          (role >)
                          (token kmalone-break-pt-season-high-at-lac)
                          (args ((stat-val 28)
                                 (histo-stat-extr {^4 ents histo-stat0-extr})))))))))))))
```

Figure 4.6: FD representation of the subnet of Fig. 4.5

4.2.1.3.4 Result streaks Similarly to historical statistics that are abstract sets of statistics, result streaks are abstract sets of games. A streak update is a second order fact relating a new game result with a streak, specifying whether the new result extends or interrupts the streak. The streak itself is a complex conceptual subnet relating multiple concepts:

- `gen-elt`: the generic element of the set, an abstract specification of the particular type of games that the streak is made of.
- `gen-elt-winner` (or `gen-elt-loser`): the winner (or loser) common to each game in the streak.
- `gen-elt-host` (or `gen-elt-visitor`): the host (or visitor) common to each game in the streak (respectively for home and road streaks; for overall streaks, this concept is not needed).
- `card`: the cardinal of the set (*i.e.,* the duration of the streak in number of games).

An example FD representing a streak extension content unit is shown in Fig. 4.7.

4.2.1.3.5 Streak records A historical streak is an abstract set of streaks, and hence an abstract set of an abstract set of games. A record streak update is thus a *third* order fact, that relates a new game result with an historical streak, specifying whether the new result extend one element of the historical streak making it the longest element of that set. An historical streak is the most complex entity of the STREAK sub-domain and involves the following relations:

- `duration`: the time span over which the set of streaks have been compiled.
- `gen-elt`: the generic element of the set, an abstract specification of the particular type of streaks that the set contains.
- `gen-elt-gen-elt`: an abstract specification of the common type of games that each streak in the set is made of.
- `gen-elt-gen-elt-winner` (or `gen-elt-gen-elt-loser`): the winner (or loser) common to each game of each streak that the set contains.
- `gen-elt-gen-elt-host` (or `gen-elt-gen-elt-visitor`): the host (or visitor) common to each game of each streak that the set contains (for sets of home or road streaks; for sets of overall streaks, this relation is not needed).
- `max-card`: the maximum cardinal for any streak in the set (*i.e.,* the length, in number of games, of the longest streak in the historical streak).

An example FD representing a streak record update content unit is shown in Fig. 4.8[3]. In this example the duration of the historical streak is indirectly defined in terms of the lifetime of the common game host of each element in the historical streak.

4.2.2 The SSS language

As explained in Section 3.3.1 and illustrated by the detailed example given in the previous section, a DSS represents both the explicit and the recoverable content of an utterance and it does so at a very fine grain. Each element in a DSS network embodies no more that the meaning of a single word or even a single semantic feature of a word. Moreover, since it simultaneously represents multiple perspectives on some concepts, only a few of these fine grained elements end up explicitly realized by a lexical item in any given utterance generated from the DSS. Finally, a DSS is flat, with no notion of constituency or dependency.

It the phrase planner that decides which elements of the DSS network to explicitly realize, how to aggregate these elements into linguistic constituents and what are the dependency relations among them. These decisions result in the SSS semantic tree. There are two distinct types of elements in such a tree:

[3] To make it fit on a single page, token features have been removed from this FD.

```
;; FD representing the content of paraphrases such as:
;; ``the Utah Jazz handed the Boston Celtics their sixth straight home defeat''
;; ``the Utah Jazz extended the Boston Celtics' losing streak to six games''

((deepsemcat entity)
 (concept game)
 (token uta-at-bos)
 (attrs ((results ((ents ((streak1 ((deepsemcat entity)
                                    (concept streak)
                                    (token bos-streak-vs-uta)
                                    (attrs ((card 6)))))
                          (streak1-gen-elt ((deepsemcat entity) (concept game)))
                          (streak1-gen-elt-host ((deepsemcat entity)
                                                 (concept team)
                                                 (token celts)
                                                 (attrs ((home "Boston") (franchise "Celtic")))))))
                   (rels ((streak1-gen-elt ((deepsemcat relation)
                                            (role gen-elt)
                                            (token bos-streak-vs-uta-gen-elt)
                                            (args ((set {^4 ents streak1})
                                                   (gen-elt {^4 ents streak1-gen-elt})))))
                          (streak1-gen-elt-loser ((deepsemcat relation)
                                                  (role loser)
                                                  (token bos-streak-vs-uta-loser)
                                                  (args ((game {^4 ents streak1-gen-elt})
                                                         (loser {^4 ents host})))))
                          (streak1-gen-elt-host ((deepsemcat relation)
                                                 (role host)
                                                 (token bos-streak-vs-uta-host)
                                                 (args ((game {^4 ents streak1-gen-elt})
                                                        (host {^4 ents host})))))
                          (streak1-ext ((deepsemcat relation)
                                        (role streak-extension)
                                        (token bos-streak-vs-uta-ext)
                                        (args ((extension top-level)
                                               (streak {^4 ents streak1})))))))))))
```

Figure 4.7: FD representing a streak extension

```
;; FD representing the record streak update fact conveyed by the uppercased constituent in the paraphrases:
;; ``the Boston Celtics' FRANCHISE RECORD six games home losing streak''
;; ``the WORST EVER sixth consecutive defeat at home for the Boston Celtics''

((deepsemcat entity)
 (concept game)
 (attrs
  ((results
    ((ents ((histo-streak1 ((deepsemcat entity) (concept histo-streak)))
            (histo-streak1-gen-elt ((deepsemcat entity) (concept streak)))
            (histo-streak1-gen-elt-gen-elt ((deepsemcat entity) (concept game)))
            (histo-streak1-extr-card ((deepsemcat entity) (concept integer)))
            (histo-streak1-gen-elt-gen-elt-host ((deepsemcat entity)
                                                 (concept team)
                                                 (attrs ((home "Boston") (franchise "Celtic")))))
            (host-lifetime ((deepsemcat entity) (concept duration)))))
     (rels ((host-lifetime ((deepsemcat relation)
                            (role lifetime)
                            (args ((entity {^4 ents host}
                                   (lifetime {^4 ents host-lifetime}))))))
            (histo-streak1-duration ((deepsemcat relation)
                                     (role duration)
                                     (args ((entity {^4 ents histo-streak1})
                                            (duration {^4 ents host-lifetime})))))
            (histo-streak1-gen-elt ((deepsemcat relation)
                                    (role gen-elt)
                                    (args ((set {^4 ents histo-streak1})
                                           (gen-elt {^4 ents histo-streak1-gen-elt})))))
            (histo-streak1-gen-elt-gen-elt ((deepsemcat relation)
                                            (role gen-elt)
                                            (args ((set {^4 ents histo-streak1-gen-elt})
                                                   (gen-elt {^4 ents histo-streak1-gen-elt-gen-elt})))))
            (histo-streak1-gen-elt-gen-elt-loser ((deepsemcat relation)
                                                  (role loser)
                                                  (args ((game {^4 ents histo-streak1-gen-elt-gen-elt})
                                                         (team {^4 ents host})))))
            (histo-streak1-gen-elt-gen-elt-host ((deepsemcat relation)
                                                 (role host)
                                                 (args ((game {^4 ents histo-streak1-gen-elt-gen-elt})
                                                        (team {^4 ents host})))))
            (histo-streak1-extr-card ((deepsemcat relation)
                                      (role max-card)
                                      (args ((card {^4 ents histo-streak1-extr-card})
                                             (set {^4 ents histo-streak1})))))
            (histo-streak1-update ((deepsemcat relation)
                                   (role >)
                                   (args ((streak-len 6)
                                          (histo-stat-extr {^4 ents histo-streak1-extr-card})))))))))))))
```

Figure 4.8: FD representing a streak record update

- *Encyclopedic* subtrees, identified by the feature (`surfsemcat encyclo`), that correspond to a whole chunk of the DSS network.
- *Rhetorical* subtrees, identified by the feature (`surfsemcat rhetor`), that are aggregates of several scattered chunks of the DSS network, linked by rhetorical relations inside the SSS.

Encyclopedic subtrees are structured by the fact generator, whereas rhetorical subtrees are structured by the phrase planner. I describe these two different types of SSS subtrees in turn in the following subsections. As opposed to a flat network, a structured tree can be trivially represented by an FD. As explained in Section B.1.3.3 of Appendix B, any FD corresponds to a directed graph. Since paths are the only features that introduce the possibility of several arcs reaching the same node, a path-free FD reduces to a tree, with each of its attributes corresponding to an arc, and each of its sub-FDs to a dependent subtree.

4.2.2.1 Encyclopedic semantic subtrees

In addition to `surfsemcat`, an encyclopedic subtree contains two more obligatory attributes:

- `concept`, the concept or role in the DSS subnet that the subtree realizes at the SSS layer.
- `onto`, specifying the ONTOlogical perspective on the concept and whose value can be either `indiv` (for INDIVidual), `event, quality, quantity, place` or `time`.

Different optional attributes are available for subtrees with different `onto` features:

- *For events*: the `args` (for ARGumentS) attributes define the thematic structure of the event; these attributes closely correspond to the DGS attributes defining the thematic structures of processes in SURGE (cf. Section B.2.2.1 of Appendix B).
- *For qualities, quantities, places and times*: the concept specific `restrictors` attributes circumscribe a specialization of the general concept indicated by the `concept` attribute.
- *For individuals*: similar `restrictors` attributes (for referring to the individual using a common NP) or the `names` attributes (for referring to the individual using a proper NP).

Finally, any encyclopedic subtree may also contain a `root` attribute, which points to either the concept or one of the optional attributes of the subtree, and defines which semantic element is to head the linguistic constituent corresponding to this subtree.

4.2.2.2 Rhetorical semantic subtrees

As explained in Section 3.3.1.2 there are three distinct types of rhetorical semantic subtrees: paratactic complexes, hypotactic complexes and rhetorical events. These three types are distinguished by the `struct` (for STRUCTure) attribute with respective values `paratax, hypotax` and `event`.

Paratactic complexes represent conjunctions and appositions and have two other attributes:

- `elts` (for ELemenTS) containing each element of the complex.
- `rel` (for RELation) specifying the semantic relation underlying the grouping of these element inside the paratactic complex and whose value in the current STREAK sub-domain can be `temporal-inclusion, teammate, co-agent, co-ref` (for CO-REFerent) or `co-occur` (for CO-OCCURrent).

Hypotactic complexes represent complex clauses or nominal and have two other top-level attributes:

- `root`, whose value is an embedded description for the head of the complex.
- `rels` (for RELationS) whose sub-attributes are the rhetorical relations linking the dependent constituents of the complex to its head; the description of each dependent constituent is embedded under one these sub-attributes.

The list of rhetorical relations currently used in the STREAK sub-domain is: **`cardinal, card-of`** (for CARDinal OF, *i.e.,* the converse of cardinal), **`compar`** (for COMPARison), **`score, co-occur, duration, standard, ordinal, opposition, result, possessor, time, location, agent-of, frequency, instrument`** and **`type`**. Most of them correspond, either directly or as converse relation, to DGS attributes used in SURGE in the description of complex nominals, clause participants, clause predicate modifiers and clause circumstancials (cf. Sections B.2 and B.3 of Appendix B).

Rhetorical events represent simple clauses whose head verb does not realize any specific element of the DSS network and also have two other top-level attributes:

- `root`: a general event class (*e.g.,* transf-poss (for TRANSFer of POSSession) for a composite material/possessive process).
- `args`: (for ARGumentS) containing the event participants cast in the argument structure corresponding to that event class indicated in the root.

These two attributes are the same as those for encyclopedic events. Finally, all three types of rhetorical subtrees can also contain a `focus` attribute containing a path indicating which one among the other attributes should appear up front in the sentence to generate.

4.2.2.3 Example FDs encoding SSSs

In Section 3.3.1.2 I gave the design level SSS for three phrases conveying the same content from different perspectives using different linguistic structures. I now give the FD encoding of each of these three SSSs. These FDs illustrate how syntagmatic paraphrasing is represented at the detailed implementation level. They also provide a variety of examples for both the encyclopedic and rhetorical subtrees discussed in the previous section.

The FD encoding of `sss1` shown on the left side of Fig. 3.5 p. 48 is given in Fig. 4.9, that of `sss2` shown on the right side of Fig. 3.5 p. 48, is given in Fig. 4.10 and that of `sss3` shown in Fig. 3.8 p. 50 is given in Fig. 4.11. Since these three examples are either hypotactically structured or event structured, an example FD encoding a paratactically structured SSS is given in Fig. 4.12.

4.2.3 The draft representation

As explained in Section 3.3.1.4, the representation of the draft in STREAK incorporates all three layers of abstraction: the DSS, the SSS and the DGS. In addition, it also contains an explicit representation of the correspondence, first between the constituents of DGS and the constituent of the SSS, and then between the constituents of the SSS and the elements of the DSS.

At the implementation level, the draft is represented by a three-layer FD with the following conventions:

- The top-level of the FD is the DGS of the draft.
- The top-level constituent of the DGS contains a special attribute `sss` whose value is the SSS for the whole draft.
- In turn, the top-level constituent of this SSS contains a special attribute `dss` whose value is the DSS for the whole draft.
- Each subconstituent C_{dgs} in the DGS also contains an `sss` attribute. As opposed to the top-level `sss` attribute whose value is an FD, the value of such an embedded `sss` attribute is a *path* pointing to the subconstituent C_{sss} in that FD, that C_{dgs} realizes at the DGS layer.
- Similarly, each subconstituent C_{sss} in the SSS contains an `dss` attribute pointing to the DSS element that C_{sss} realizes at the SSS layer.

```
FD representing the SSS for paraphrases such as:
``Orlando defeated Toronto at home''
``Orlando triumphed over Toronto at home''

((surfsemcat rhetor)
 (struct hypotax)
 (root ((surfsemcat encyclo)
        (onto event)
        (concept beat)
        (args ((agent ((surfsemcat encyclo)
                       (onto indiv)
                       (concept team)
                       (names ((home "Orlando")))
                       (root {^ names})))
               (affected ((surfsemcat encyclo)
                          (onto indiv)
                          (concept team)
                          (names ((home "Toronto")))
                          (root {^ names})))))
        (focus {^ args agent})))
 (rels ((location ((surfsemcat encyclo) (onto place) (concept host)))))
 (focus {^ root}))
```

Figure 4.9: FD encoding of `sss1`

```
FD representing the SSS for paraphrases such as:
``The Magic's home victory over the Raptors''
``The homecourt win of the Magic against the Raptors''

((surfsemcat rhetor)
 (struct hypotax)
 (root ((surfsemcat encyclo) (onto indiv) (concept winner)))
 (rels ((possessor ((surfsemcat encyclo) (onto indiv) (concept team) (names ((franchise "Magic")))))
        (location ((surfsemcat encyclo) (onto indiv) (concept host)))
        (opposition ((surfsemcat encyclo) (onto indiv) (concept team) (names ((franchise "Raptor"))))))))
```

Figure 4.10: FD encoding of `sss2`

```
FD representing the SSS for paraphrases such as:
``The Magic claimed a home win over the Raptors''
``The Magic posted a homecourt victory against the Raptors''

((surfsemcat rhetor)
 (struct event)
 (root activity)
 (args ((agent ((surfsemcat encyclo)
                (onto indiv)
                (concept team)
                (names ((franchise "Magic")))
                (root {^ names})))
        (range ((surfsemcat rhetor)
                (struct hypotax)
                (root ((surfsemcat encyclo) (onto indiv) (concept winner)))
                (rels ((location ((surfsemcat encyclo) (onto indiv) (concept host)))
                       (opposition ((surfsemcat encyclo)
                                    (onto indiv)
                                    (concept team)
                                    (names ((franchise "Raptor")))
                                    (root {^ names})))))))))
 (focus {^ args agent}))
```

Figure 4.11: FD encoding of `sss3`

In Fig. 3.12 of Section 3.3.1.4 I gave a design level, graph representation of the draft for the phrase "the Orlando Magic defeated the Toronto Raptors". In Fig. 4.13, I give the corresponding implementation level FD representation of the same draft. The `sss` and `dss` features are emphasized in uppercase.

4.3 The processing components

Aside from the portable syntactic front-end, SURGE, STREAK proper is made of three components: the phrase planner, the lexicalizer and the reviser. The design and task of these three modules were described in Section 3.3.2. All three modules were implemented declaratively in FUF. The first two modules use the top-down recursive unification mechanism of FUF as a rule interpreter. The set of phrase planning rules and the set of lexicalization rules are factorized into two hierarchies, each represented as a Fuf Grammar (FG). In such a hierarchy, several rules sharing a common part are grouped in a FUF conjunction in which the top-level features encode the common part and are followed by a FUF disjunction encoding the distinct parts, proper to each rule. The reviser also consists of an interpreter and a declarative rule base. However, since the application of revision rule is in general a non-monotonic operation, the revision rule interpreter is far more complex than the phrase planning and lexicalization rule interpreters and does not entirely rely on unification. It also relies on an array of low-level FD manipulation functions provided by the FUF package. I present the implementation of the three components in detail in the following subsections.

4.3.1 The phrase planner

The task of the phrase planner is to map an input DSS onto an output SSS. As explained in Section 3.3.2, this task involves:

- Picking which element in the DSS network to realize explicitly in a word of the sentence to generate (and thus which to leave implicit, recoverable from either the sentence structure or knowledge about

```
FD representing the SSS for paraphrases such as:
``Barkley scored 28 points and Majerle added 24''
``Barkley struck for 28 points and Majerle fired in 24''

((surfsemcat rhetor)
 (struct paratax)
 (elts ((1 ((surfsemcat encyclo)
           (onto event)
           (concept game-stat-rel)
           (args ((agent ((surfsemcat encyclo)
                          (onto indiv)
                          (concept player)
                          (names ((last-name "Barkley")))
                          (root {^ names})))
                  (created ((surfsemcat encyclo)
                            (onto quantity)
                            (concept game-stat)
                            (restrictors ((value 28) (unit pts)))
                            (root {^ restrictors})))))))
        (2 ((surfsemcat encyclo)
           (onto event)
           (concept game-stat-rel)
           (args ((agent ((surfsemcat encyclo)
                          (onto indiv)
                          (concept player)
                          (names ((last-name "Majerle")))
                          (root {^ names})))
                  (created ((surfsemcat encyclo)
                            (onto quantity)
                            (concept game-stat)
                            (restrictors ((value 24)))
                            (root {^ restrictors}))))))))))
```

Figure 4.12: Example FD encoding a paratactically structured SSS

the domain).

- Choosing an ontological perspective from which to present each picked element.
- Grouping the picked element into constituents.
- Choosing rhetorical relations linking the picked elements inside each constituent.

The input to the phrase planner consists of an FD representing a DSS. This FD is embedded under a `dss` attribute and the result is then unified with the FG representing the phrase planning rule base. The output is an enriched FD containing, at its top-level, the SSS corresponding to the input DSS. A round of phrase planning thus simply consists of a single top-down recursive unification operation. Remember however, that the phrase planner is called several times during the generation of a sentence, once to plan the initial draft sentence and then repeatedly to plan each phrase incorporated to the draft during revision.

It is the output SSS layer that is built top-down during unification. The top-level SSS constituent is built first. This task includes translating the features of the DSS element chosen as the SSS head and then casting other related DSS elements inside the subconstituent structure of this top-level SSS. Once this is done, recursion starts on each of these subconstituent.

```
((cat clause)
 (tense past)
 (process ((type material) (lex "defeat")))
 (partic ((agent ((cat proper)
                  (head ((cat team-name)
                         (franchise ((lex "Magic")))
                         (home ((lex "Orlando")))))
                  (SSS {^3 SSS ARGS AGENT})))
          (affected ((cat proper)
                     (head ((cat team-name)
                            (franchise ((lex "Raptor")))
                            (home ((lex "Toronto")))))
                     (SSS {^3 SSS ARGS AFFECTED})))))
 (SSS ((surfsemcat encyclo)
       (onto event)
       (concept beat)
       (args ((agent ((surfsemcat encyclo)
                      (onto indiv)
                      (concept team)
                      (names ((home "Orlando")))
                      (root {^ names})
                      (DSS {^3 DSS RELS RESULT ARGS WINNER})))
              (affected ((surfsemcat encyclo)
                         (onto indiv)
                         (concept team)
                         (names ((home "Toronto")))
                         (root {^ names})
                         (DSS {^3 DSS RELS RESULT ARGS LOSER})))))
       (focus {^ args agent})
       (DSS ((ents ((host ((deepsemcat entity)
                           (concept team) (token magic)
                           (attrs ((home "Orlando") (franchise "Magic")))))
                    (visitor ((deepsemcat entity)
                              (concept team) (token raptors)
                              (attrs ((home "Toronto") (franchise "Raptor")))))))
             (rels ((winner ((deepsemcat relation)
                             (role winner) (token tor-at-orl-winner)
                             (args ((game top-level) (winner {^4 ents host})))))
                    (loser ((deepsemcat relation)
                            (role loser) (token tor-at-orl-loser)
                            (args ((game top-level) (loser {^4 ents visitor})))))
                    (result ((deepsemcat relation)
                             (role beat) (token tor-at-orl-result)
                             (args ((winner {^4 ents host}) (loser {^4 ents visitor}))))))))))))
```

Figure 4.13: FD representing the draft phrase “the Orlando Magic defeated the Toronto Raptors”

```
(SSS ((surfsemcat encyclo)
      (onto event)
      (concept beat)
      (focus {^ args agent})
      (args ((agent ((DSS {^3 DSS ATTRS RELS RESULT ARGS WINNER})))
     (affected ((DSS {^3 DSS ATTRS RELS RESULT ARGS LOSER})))))
      (DEEPMAP-CSET ((+ {^ args agent} {^ args affected})))
      (DSS ... cf. Fig. 4.13, p.78 )))
```

Figure 4.14: Partial SSS after top-level unification

To illustrate how an SSS layer is built on an example, consider again the SSS layer of the draft FD in Fig. 4.13. It results from the recursive unification of the DSS layer in the same draft FD, with the FG embodying the phrase planning rules. The partial SSS obtained after the top-level unification is shown in Fig. 4.14.

At this point the following decisions have been made:

- Out of the five elements in the DSS network, only three will be explicitly realized by a linguistic element in the sentence generate: the two entities `host` and `visitor` and the relation `result`.
- This `result` relation will head the linguistic realization.
- It will be presented as a material event.
- The `winner` argument of this relation (which in this particular case is also the `host`) will be mapped onto the agent of the material event (this is indicated by the DSS pointer filling `agent`)
- The `loser` argument of this relation (which in this particular case is also the `visitor`) will be mapped onto the agent of the material event (this is indicated by the DSS pointer filling `affected`)

The `deepmap-cset` meta-attribute indicates that first the sub-FD under `agent`, and then the sub-FD under `affected` must be in turn unified with the phrase planning FG. After both these unifications are completed, the resulting SSS is the final one shown in Fig. 4.13.

The phrase planning rules relevant for the top-level mapping stage of the example SSS above are shown in Fig. 4.15. They are encoded as the FG conjunction `game-result-map`[4].

This conjunction encodes three alternative rules for planning the game result. There are two main parts in this conjunction. The first part is the common precondition of these three rules testing the input DSS. This precondition is encoded by the subconjunction `game-result` at the bottom of the figure. This subconjunction is a DSS pattern which matches any subnet representing a game result. The second part contains the different SSS output building actions proper to each of these three rules.

Rule 1 encodes the realization of the game result as a clause headed by a full verb. This full verb conveys the `result` relation of the DSS. The output of Fig. 4.14 results from the application of rule 1. Rule 2 encodes the realization of the game result as a clause headed by a support verb. This support verb does not convey in itself any element in the DSS, but only serves as a syntactic support for its range argument that realizes the `winner` relation of the game result. The choice between the rules 1 and 2 is random. It encodes the paraphrasing alternative between:

- "the Orlando Magic defeated the Toronto Raptors" (Rule 1), and
- "the Orlando Magic claimed a victory over the Toronto Raptors" (Rule 2)

[4] This conjunction is in fact a simplified version of the one actually used in the code, which also maps the final score-line of the game - ignored here - as part of the game result.

```
(def-conj game-result-map
  (dss ((:& game-result)))
  (ALT full-result
    (((already-mapped none)
      (RALT full-result-struct
        (
         ;; Game result planning rule 1: clauses headed by full verb,
         ;; e.g., ``WINNER defeated LOSER''
         (surfsemcat encyclo)
         (onto event)
         (concept {^ dss rels result role})
         (args ((agent ((dss {^3 dss rels result args winner})))
                (affected ((dss {^3 dss rels result args loser})))))
         (deepmap-CSET ((= {^ args agent} {^ args affected}))))

         ;; Game result planning rule 2: clauses headed by support verb,
         ;; e.g., ``WINNER claimed <a victory over LOSER> (recurse for <> part)
         ((surfsemcat rhetor)
          (struct event)
          (root activity)
          (args ((agent ((dss {^3 dss rels winner args winner})))
                 (range ((dss {^3 dss})
                         (already-mapped winner)))))
          (deepmap-CSET ((= {^ args agent} {^ args range})))))))
     (
      ;; Game result planning rule 3: NP e.g., ``a victory over LOSER''
      (already-mapped #(under winner))
      (surfsemcat rhetor)
      (struct hypotax)
      (focus {^ rels score})
      (root ((dss {^2 dss rels winner})))
      (rels ((opposition ((dss {^3 dss rels loser args loser})))))
      (deepmap-CSET ((= {^ root} {^ rels opposition}))))))


(def-conj game-result
  (ents ((host ((deepsemcat #(under entity)) (concept #(under team))))
         (visitor ((deepsemcat #(under entity)) (concept #(under team))))))
  (rels ((winner ((deepsemcat #(under relation))
                  (role #(under winner))
                  (args ((game GIVEN) (winner GIVEN)))))
         (loser ((deepsemcat #(under relation))
                 (role #(under loser))
                 (args ((game GIVEN) (loser GIVEN)))))
         (result ((deepsemcat #(under relation))
                  (role #(under beat))
                  (args ((winner GIVEN) (loser GIVEN))))))))
```

Figure 4.15: Example of phrase planning rules

There are many such random choices in the phrase planner. They encode options that are, in most circumstances, equivalent. Their presence insures that the generator will not always produce the same output linguistic form when repeatedly given in input the same type of information to report. In order to force the choice of one of these random alternatives, a partial description of the desired output can be passed as input to the phrase planner as an optional parameter. Only the options that can unify with this partial output can then be chosen in each disjunction of the phrase planning FG.

In the top-down process of planning an entire draft, there are cases when the winning team has already been mapped inside a matrix SSS constituent before the realization of the game result starts. In such contexts, none of the two forms above can be used. This the case for example, when, at the time of recursing on the game result subnet, the phrase planner has already built the SSS for a top-level draft such as (1) below. In this draft form the use of a material/locative event to link the main statistic to the game result instead of a temporal relation as in (2) or a paratactic complex as in (3) prohibits the application of rules 1 and 2 for planning the structure of the game result itself.

1. "Shaquille O'Neal scored 39 points leading the Orlando Magic to ...".
2. "Shaquille O'Neal scored 39 points as the Orlando Magic defeated the Toronto Raptors"
3. "Shaquille O'Neal scored 39 points and the Orlando Magic claimed a victory over the Toronto Raptors"

Rule 3 handles these cases. It realizes the game result as an NP such as "a victory over the Toronto Raptors" which does not mention the winning team. The feature `already-mapped` tests whether, at the moment of mapping the game result subnet, the winning team has or not already been mapped in the matrix SSS constituent. Since this NP realization of the game result perfectly fits the pattern of the range argument of the support verb clause realization, this range argument is built through recursion within this very FG conjunction.

4.3.2 The lexicalizer

The task of the lexicalizer is to map an input SSS onto an output DGS. As explained in Section 3.3.2, this task involves:

- Choosing, for each semantic constituent in the SSS, the syntactic category of the linguistic constituent realizing it at the DGS layer.
- Choosing the open-class lexical items of this linguistic constituent.
- Specifying the non-default values for the syntactic features associated with the chosen syntactic category.
- Mapping the rhetorical relations linking the semantic constituents at the SSS layer onto the thematic and syntactic roles linking the corresponding syntactic constituents at the DGS layer.

The phrase planner and the lexicalizer share the same interpreter, but each uses it with a different rule base. Lexicalization is therefore implemented as the top-down recursive unification of an FD, where the input SSS has been embedded under an `sss` attribute, with an FG encoding the lexicalization rule base.

For a linguistic constituent C_{dgs} realizing at the DGS layer an SSS subtree C_{sss}, the choice of syntactic category depends essentially on the `onto` feature of C_{sss} and the choice of lexical head depends essentially on the `concept` feature of C_{sss}. This is true for both simple and complex linguistic constituents[5]. However, at the SSS layer, the latter are represented by rhetorical subtrees, lacking these two features - `onto` and `concept` - that are proper to encyclopedic subtrees (as explained in Section 4.2.2). These rhetorical subtrees are recursive structures whose `root` attribute can be filled either by an encyclopedic subtrees (end of recursion) or by another rhetorical subtree (recursion). For each syntactic constituent, the first task of the lexicalizer

[5] Only *hypotactic* complexes are considered here, since the paratactic ones do not have a lexical head (and those of type `list` do not even have a syntactic category).

is to fill an attribute `sss-root` which points to the head *encyclopedic* element of the input SSS subtree. For encyclopedic subtrees the value of this attribute is simply `sss`. But for rhetorical ones it can be a longer path following several `root` indirections down the recursive rhetorical structure. For example, when given as input the SSS at the top of Fig. 4.11 p. 76, the lexicalizer would add a pair `(sss-root {^ sss root})`. By making the relevant encyclopedic features accessible at the top-level, this `sss-root` feature allows the choice of syntactic category and lexical head to be handled uniformly regardless of the input's rhetorical complexity.

To illustrate the process of lexical choice in general let us consider verb choice. Excerpts from the FUF disjunction encoding verb choice rules in STREAK's lexicalizing FG are shown in Fig. 4.16. A first high level look at this disjunction reveals several points about lexical choice:

- In general, verbs (at least finite ones) are appropriate to realize the head constituent of event descriptions. This is indicated by the fact that the testing part of each branch is under `sss-root` and that all branches correspond to one of the two types of event descriptions at the SSS layer: rhetorical (branch 1) and encyclopedic (branch 2 and 3).
- The choice of verb is first indexed by either the `concept` feature (for encyclopedic events) or the general event type (specified by the `root` feature in rhetorical events) that they can realize. A polysemous verb can thus appear in as many different branches as it has senses.
- Each bottom branch represents a set of synonymous verbs that are randomly chosen.
- Single word verbs and prepositional verbs are uniformly represented.

The first two branches of the disjunction encodes some of the head verbs' options for clauses realizing game results. They are the options involved for the type of game result input SSS built by the application of the phrase planning rules discussed in the previous section. Each fits only a given syntactic structure: the first branch fits support verb headed clauses with agent and range participants as in:
"Orlando posted a victory over Toronto",

while the second branch fits full verb headed clauses with agent and affected participants, as in:
"Orlando routed Toronto".

This type of syntactic structure constraints on lexical choice underlines the dual aspects of lexicalization:

- The *syntagmatic* aspect consisting of selecting a word such that the constituents of the semantic input can be cast in the argument structure of the word.
- The *paradigmatic* aspect consisting of selecting a word that can realize the head concept in the semantic input.

In STREAK these two aspects are separated in different disjunctions of the lexicalization FG. The `surfmap-verbs` disjunction shown in Fig. 4.16 only encodes the paradigmatic aspect of verb choice. The syntagmatic aspect is encoded in the `surfmap-partic` disjunction.

The third branch in `surfmap-verbs` encodes the choice of verbs for realizing game statistics. It illustrates how constraints other than the root concept and the syntactic structure influence lexical choice. The first of these constraints, encoded in the `stat-num` feature, is passed to the lexicalizer by the reviser. It tests whether the game statistic clause under construction is the first among the additional statistics added to the the main statistic during the revision of the draft. It indicates that in such a case the most appropriate verb to choose is "to add" as in: "Charles Barkley scored 29 points and Dan Majerle **added** 24".

How such constraints are passed from the reviser to either the phrase planner or the lexicalizer is explained in the next section.

The rest of the disjunction distinguishes between versatile verbs (*e.g., "to have"*) valid for lexicalizing any type of game statistic as shown by:
"Barkley had 29 points" and "Barkley had 20 rebounds".

```
(def-alt surfmap-verbs

  ...

  ;; activity (i.e. verbs part of collocation with range object)
  ((sss-root ((root #(under activity))
      (struct #(under event))))
   (proc ((lex ((RALT ("claim" "record" "post" "pull out" "clinch" "nail down")))))))

  ...

  ;; beat
  ((sss-root ((concept #(under beat))
      (onto #(under event))))
   (proc ((lex ((RALT ("defeat" "beat" "triumph over" "coast past" "down" "rout")))))))

  ;; game statistic
  ((sss-root ((concept #(under game-stat-rel))
      (onto #(under event))
      (args ((created ((concept #(under game-stat)))))))
   (ALT stat-num
     (
      ;; For 1st additional statistic, use "to add"
      ((stat-num #(under 1)) (proc ((lex "add"))))

      ;; Versatile verbs for any type of statistics
      ((RALT game-stat-rel-v-specificity
 (((proc ((lex ((RALT ("have" "finish with" "wind up with" "collect" ... )))))))
  ;; Verbs specialized by statistic type
  ((ALT stat-unit
             (((sss-root ((args ((created ((restrictors ((unit #(under pt))))))))))
       (proc ((lex ((RALT ("score" "net" "pump in" "fire in" "strike for" ...)))))))

      ...

      ((sss-root ((args ((created ((restrictors ((unit #(under reb))))))))))
       (proc ((lex ((RALT ("grab" "haul in" "get" "snare" "pull down" ... ))))))))))))))))

  ...

)
```

Figure 4.16: Encoding verb choice in STREAK

from specialized ones valid for only one specific type of statistics as shown by:
"Barkley fired in 29 points" and "Barkley pulled down 20 points" while
? "Barkley pulled down 29 points" and ? "Barkley fired in 20 points".

4.3.3 The reviser

The input to the reviser is twofold:

- a three-layer FD encoding the initial draft at the three layers of abstraction (DSS, SSS and DGS) such as the example given in Fig. 4.13.
- A list of FDs, each encoding a DSS subnet representing a floating fact to opportunistically incorporate to the draft, in order of decreasing relevance.

The output of the reviser is a three-layer FD encoding the final draft which incorporates the content conveyed by the initial draft plus as many floating facts as could be added without exceeding a lexical length of 45 and a syntactic depth of 10 for the final draft.

The reviser consists of three parts:

- The declarative revision rule base encoding the revision operations presented in Section 2.5.
- The revision rule interpreter that takes as input both the current draft and *a single* floating fact and performs *one* revision increment.
- The revision monitor that repeatedly calls the revision rule interpreter and controls the overall incremental revision process.

I discuss each of these three parts in turn in the following subsections.

4.3.3.1 The revision rule base

A revision rule has two parts: a Left Hand Side (LHS) which specifies the conditions in which the rule can apply and a Right Hand Side (RHS) which specifies the set of transformations that the draft undergoes during the application of the rule. It is thus encoded as an FD with two top-level attributes: `lhs` and `rhs`. The revision rule base is encoded as an FG where revision rules are factorized in a recursive set of FUF disjunctions and conjunctions. The resulting FG structure parallels the hierarchical structure of the revision operations presented in Section 2.5. The top-level disjunction distinguishes among the general classes of revision tools, such as `adjoin`, `absorb` or `adjunctization`, and the embedded disjunctions encode the lower-level distinctions in terms of the accompanying side transformations, the semantic and/or syntactic roles of the constituent added, displaced or deleted from the draft by the operation, etc.

The LHS of a revision rule is encoded as an FD with three attributes:

- `bls` (Base Layered Specification), whose value is a three-layer FD defining the type of drafts onto which the rule can be applied.
- `adss` (Additional Deep Semantic Specification), whose value is a DSS FD defining the type of floating facts that the rule can incorporate to the draft.
- `tool`, whose value is the name of the revision operation encoded by the rule.

This last attribute is optional and is used only for control and testing purposes. When it is not present, the reviser uses the first rule whose `bls` feature matches the current draft and whose `adss` matches the first DSS in the ordered floating fact list. Each revision operation, whether general or specific, is given a name. These names are grouped into a hierarchy paralleling the specialization hierarchy of the corresponding operations using the `define-feature-type` construct of FUF.

The RHS of a revision rule simply consists of a list of *revision actions*. These revision actions are the building blocks used to implement the structural transformations involved in the application of the revision rules. The *revision workspace* onto which each of these actions operate is an FD with four top-level attributes:

- `bls` (Base Layered Specification) whose value is the three-layer FD encoding the old draft before the application of any RHS action.
- `rls` (Revised Layered Specification) whose value is the three-layer FD encoding the new draft under construction and resulting from the RHS actions applied so far; initially it is a copy of `bls`.
- `dss` whose value is an FD encoding the common DSS layer of the `bls` and the `rls` throughout the application of the actions.
- `adss` (Addition Deep Semantic Specification) whose value is the DSS FD encoding the floating fact to incorporate to the draft for the current revision increment.

Two points concerning the structure of this revision workspace deserve explanations. The first points is the need for keeping the original input draft around (in the `bls`) during the application of the actions building the new draft (in the `rls`). This is necessary because, as explained in Section 2.5, most revision operations are *non*-monotonic: in order to accommodate a new floating fact, they alter the linguistic realization of the draft content. Such an alteration is decomposed into several low-level actions cutting and pasting the linguistic constituents of the draft. The `bls` buffers the cut elements before they get pasted in a new location in the `rls`.

The second point requiring explanation is the fact that the the `bls` and the `rls` share a common DSS layer throughout the application of the revision actions. This is the case because these actions are implementing the accommodation of the new fact (encoded in the `adss`) *only* at the SSS and DGS layers. As mentioned in Section 4.2.1.2, incorporating this new fact to the draft at the DSS layer only involves unifying the `adss` with the DSS layer of the input draft. The result of this unification is then placed in the top-level `dss` attribute during the initialization of the revision workspace. To avoid duplication of content, which may introduce inconsistency in such a non-monotonic context, the embedded `dss` attributes inside the two three-layer FDs (the `bls` and the `rls`) then becomes path to the top-level `dss` attribute. Each element in the `adss` also becomes a pointer to the corresponding element under the top-level `dss` attribute. Thus, before any action is applied to the draft, its DSS layer has already being revised. These different mechanisms for implementing revision at the DSS layer on the one hand (unification) and at the SSS and DGS layers on the other (revision rule application) is motivated by the fact that the elements of a DSS are *not* grouped into constituents, whereas the elements of the SSS and DGS are. Therefore, the addition of a new element at the DSS layer is necessarily a *global* operation, whereas at the SSS and DGS layer it is best viewed as *local* to a particular constituent. I come back to this last point in the next section while describing the revision rule interpreter.

There are five types of RHS actions acting upon the revision workspace:

- `(add-fd fd address)`: inserts `fd` as a sub-FD under the path `address` inside the workspace FD; `fd` can be either an atom or a structured FD but *not* a path.
- `(add-path path address)`: inserts `path` as a sub-FD inside the workspace FD under the path `address`.
- `(cp-fd input-address output-address)`[6]: relocates the sub-FD located under the path `input-address` in the workspace FD and inserts that copy under the path `output-address`.
- `(del-fd address)`: DELetes the sub-FD under the path `address` in the workspace FD.
- `(map-fd input-address output-address mapping-type mapping-constraint)`:
  1. Relocates the sub-FD located under the path `input-address` in the workspace FD.
  2. Calls either the phrase planner (if `mapping-type` = `deep`) or the lexicalizer (if `mapping-type` = `surf`) on that copy.

[6] cp-fd stands for CoPy FD.

```
 0 (def-conj nominalization
 1   (lhs ((bls ((:& material-basic-res-cluster)))
 2         (adss ((attrs ((results ((:& los-streak-ext)))))))
 3         (tool nominalization)))
 4   (rhs ((del-fd {})
 5         (add-fd ((surfsemcat rhetor)
 6                  (struct hypotax)
 7                  (root ((surfsemcat rhetor) (struct event) (root transf-poss))))
 8                  {sss})
 9         (map-fd {rls sss} {} surf ((fills time-rel)))
10         (add-path {rls partic affected} {partic possessor})
11         (cp-fd {bls sss rels score} {sss rels score})
12         (cp-fd {bls pred-modif score} {pred-modif score})
13         (cp-fd {bls sss-root args agent} {sss-root args agent})
14         (cp-fd {bls partic agent} {partic agent})
15         (cp-fd {bls sss-root args affected} {sss-root args affected})
16         (cp-fd {bls partic affected} {partic affected})
17         (map-fd {adss attrs results}
18                 {sss-root args possessed}
19                 deep
20                 ((root ((onto indiv)))))
21         (map-fd {rls sss-root args possessed} {partic possessed} surf))))
```

Figure 4.17: Encoding the nominalization revision tool in STREAK

3. Inserts the enriched FD resulting from this call under the path `output-address` in the workspace FD.
4. Replaces the input to the phrase planner (respectively lexicalizer) filling the `dss` (respectively `sss`) attribute in the enriched FD by a path back-pointing to the input address from which it was originally copied (in order to avoid the introduction of duplicates in the workspace).

The insertions and relocations performed by the actions above are implemented by calling the special extension of FUF for non-monotonic processing of FDs presented in Section B.1.3.7 of Appendix B. To illustrate the implementation of revision rules as FDs, the FUF conjunction encoding the nominalization revision rules is given in Fig. 4.17.

The `lhs` of this conjunction (lines 1-3) specifies that nominalization can be used to:

- Add a floating fact of type losing streak extension (the pattern for this type of additional content in given in the `los-streak-ext` conjunction at the bottom of Fig. 4.18, lines 22-40).
- Add such fact onto a draft clause of type material which no other floating fact has yet been attached (the pattern for this type of draft subconstituent is given in the `material-basic-res-cluster` (material BASIC game RESult CLUSTER) at the top of Fig. 4.18).

At the beginning of the revision increment, the `rls` for the whole draft is initialized as a copy of the corresponding `bls`. The first action of any non-monotonic revision is thus to delete the matching draft subconstituent which is, in the case of nominalization, the full verb clause to be replaced by a support verb clause. This is done by the action `del-fd` on line 4. For example this action would result in the transformation of the sentence:

"O'Neal scored 39 points as	Orlando	defeated	Toronto	99-92"
Main Statistic Cluster	*Agent*	*Full Verb*	*Affected*	*Score*

into: "O'Neal scored 39 points as ... "

```
(def-conj material-basic-res-cluster
  (cat #(under clause))
  (partic ((agent ((sss {^3 sss root args agent})))
           (affected ((sss {^3 sss root args affected})))))
  (pred-modif ((score ((sss {^3 sss rels score})))))
  (sss ((surfsemcat #(under rhetor))
        (struct #(under hypotax))
        (root ((surfsemcat #(under encyclo))
               (args ((agent ((surfsemcat #(under encyclo))
                              (dss {^3 dss args winner})))
                      (affected ((surfsemcat #(under encyclo))
                                 (dss {^3 dss args loser})))))
               (dss ((deepsemcat #(under relation))
                     (role #(under beat))
                     (args ((winner ((deepsemcat #(under entity))
                                     (concept #(under team))))
                            (loser ((deepsemcat #(under entity))
                                    (concept #(under team)))))))))))
        (rels ((score ((surfsemcat #(under encyclo))
                       (dss ((deepsemcat #(under entity))
                             (concept #(under score))))))))))

(def-conj los-streak-ext
  (:& streak-ext)
  (rels ((streak1-gen-elt-loser ((deepsemcat #(under relation))
                                 (role #(under loser)))))))

(def-conj streak-ext
  (ents ((streak1 ((:& streak)))
         (streak1-gen-elt ((deepsemcat #(under entity))
                           (concept #(under game))))))
  (rels ((streak1-gen-elt ((deepsemcat #(under relation))
                           (role #(under gen-elt))
                           (args ((set {^4 ents streak1})
                                  (gen-elt {^4 ents streak1-gen-elt})))))
         (streak1-ext ((deepsemcat #(under relation))
                       (role #(under streak-extension))
                       (args ((extension #(under top-level))
                              (streak {^4 ents streak1}))))))))

(def-conj streak (deepsemcat #(under entity))
                 (concept #(under streak))
                 (attrs ((card GIVEN))))
```

Figure 4.18: Testing game result material clauses and losing streak extensions

The actions immediately following `del-fd`, involves building the top-level of the replacement clause. The `add-fd` on lines 5-8 first specifies a general transfer of possession event at the SSS layer. The `map-fd` on line 9 then calls lexicalizer to map this structure at the DGS layer. The FD `((fills time-rels))` is passed as a constraint indicating the relevant lexicalization context, cases such as this in which the clause structure to build is to appear as a dependent temporal clause. This prevents the lexicalizer from picking a mood incompatible with this dependency context. This call to the lexicalizer and the following `add-path` action result in the insertion of a composite material/possessive clause structure headed by a compatible support verb (*e.g.,* "to hand").

At this point the draft has become:

"O'Neal scored 39 points as	...	handed	...	...	..."
Main Statistic Cluster	*Agent*	*Support Verb*	*Affected-Possessor*	*Possessed*	*Score*

The `cp-fd` actions on lines 11-16 copy the subconstituents of the initial clauses that are not left unchanged by the nominalization. At this point the draft has become:

"O'Neal scored 39 points as	Orlando	handed	Toronto	...	99-92".
Main Statistic Cluster	*Agent*	*Support Verb*	*Affected/Possessor*	*Possessed*	*Score*

The `map-fd` action on lines 17-20 calls the phrase planner to build the internal structure of the NP expressing the losing streak fact to add to the draft. The FD `((root ((onto indiv))))` is passed as a constraint indicating to the planner which ontological perspective on this losing streak is appropriate in the context of this revision. In this case, since the streak must be realized by a nominal, the right perspective is individual. The final `map-fd` action on line 21 calls the lexicalizer on the SSS that the planner just built. The result is a DGS for a nominal expressing the additional streak fact and which fills the new `possessed` role in the revised clause structure.

The nominalization has now been completed, resulting in the final draft:

"O'Neal scored 39 points as	Orlando	handed	Toronto	their 10th straight defeat	99-92".
Main Statistic Cluster	*Agent*	*Support Verb*	*Affected/Possessor*	*Possessed*	*Score*

This revision rule example illustrates the fact that the `map-fd` action allows revision rules to remain general, indicating only *where* and *how* to attach the phrase realizing the new floating fact inside the draft. The internal content organization, wording and syntactic form of this new phrase is chosen by the phrase planner and the lexicalizer in the context of the revision, when called via a `map-fd` action.

4.3.3.2 The revision rule interpreter

The task of the revision rule interpreter is to perform one increment of revision. It thus takes as input a three-layer FD encoding the current draft (called `bls` for Base Layered Specification) and a DSS FD encoding the new fact to incorporate to the draft (called `adss` for Additional Deep Semantic Specification).

The incorporation is performed in two stages: (1) triggering a revision rule and (2) applying it to the draft. I discuss these two stages in the following two subsections.

4.3.3.2.1 Triggering a revision rule To insure the scalability of the reviser, the set of revision rules must be as general as possible. They must be as abstract as the revision operations they implement. As we have seen above, one way this is achieved is by encoding in the revision rules only the attachment point and method for the new fact and relying on independent modules (the phrase planner and lexicalizer) for the realization of the new fact as a new linguistic constituent. This way, the same revision rule, for example `adjoin of classifier into nominal`, can be used for revision increments such as:

- "Malone scored 39 points and Stockton dished out 29 assists as Utah defeated Denver 99-91" (initial draft)
- "Malone scored **a season high** 39 points and Stockton dished out 29 assists as Utah defeated Denver 99-91" (first revision increment)
- "Malone scored a season high 39 points and Stockton dished out **a franchise record** 25 assists as Utah defeated Denver 99-91" (second revision increment)

even though for each increment the meaning and wording of the added phrase (in bold) is different.

Another way to keep revision rules general is to have them specify only the *type* of constituents onto which a given type of new fact can be attached, independently of the *location* where these constituents may appear inside the linguistic structure of the draft. This is again illustrated by the example above where at each round the same revision rule is applied, but at different locations inside the draft structure.

Relying on revision rules whose semantics are *local* is desirably modular but come with a cost: triggering a revision rule does not simply involve searching the rule base for an appropriate rule but also searching the draft linguistic structure for a constituent onto which to apply the rule.

The outer loop of the revision rule interpreter thus consists of a top-down, depth-first traversal of the input draft at the DGS layer. For each constituent C encountered during this traversal, the interpreter builds a *revision input*, an FD with three top-level attributes:

- `bls` whose value is a three-layer FD encoding the DGS, SSS and DSS of C.
- `adss` whose value is the DSS FD encoding the new fact to incorporate to the draft.
- `tool` whose value is the name of a specific revision operation to use for the increment (if such a restriction was passed to the interpreter input as an optional input parameter).

The format of this revision input matches that of a revision rule LHS (given in the previous section). Searching for an appropriate rule is thus performed by unifying such a revision input (embedded under the attribute `lhs`) with the revision rule base for each constituent encountered during the traversal of the draft linguistic structure. When this unification fails it means that there are no revision rule available to incorporate the new fact to the particular constituent C reached at this point in the draft traversal. The revision rule interpreter then recurses on the subconstituents of C (at the DGS layer) building a new revision input for each of them. When this unification succeeds, it results is an *instantiated revision rule*, whose LHS matches the revision input and whose RHS contains the revision actions to apply to C. The control regime of this search is simple: the first constituent onto which the new fact can be incorporated is always chosen. Similarly, the first revision rule whose LHS matches the revision input for a given draft constituent is also always chosen. If there is no match between any of the revision inputs built for each draft DGS constituent and any revision rule LHS, the input floating fact cannot be added and the revision interpreter returns to the revision monitor the input draft unchanged.

Consider the first revision in the example above with the tool `adjoin of classifier into nominal` specified in the input. The draft traversal will successively attempt to apply this tool onto the following constituents:

1. "Malone scored 39 points and Stockton dished out 29 assists as Utah defeated Denver 99-91"
2. "Malone scored 39 points and Stockton dished out 29 assists"
3. "Malone scored 39 points"
4. "Malone"
5. "scored"
6. "39 points"

For attempts 1-3 and 5, unification with the revision rule base fails because the draft constituent is not of the syntactic category specified in the input tool: nominal. The fourth attempt fails due to a semantic constraint: the property of a given entity can be only attached to nominals referring to that very entity. In the example at hand, the record nature of a statistic can only be attached to a nominal referring to that statistic. It cannot be attached to a nominal referring to a player, even whose performance is summed-up by that statistic, as shown by:
? "A season high Malone scored 39 points."

4.3.3.2.2 Applying a revision rule The application of the instantiated rule starts with the building of the revision workspace. In Section 4.3.3.1 we defined this workspace as an FD with four top-level attribute: **bls, rls, dss** and **adss**. Since the application of the instantiated rule must be *local* to the draft subconstituent C that triggered the rule during the top-down traversal of the draft, the **bls** attribute must be initialized as an three-layer FD representing C at all three layers of abstraction. The **rls** attribute can then be initialized as a copy of this local **bls**. The **dss** attribute must remain *global* (*i.e.,* for the whole draft) however, since as explained in Section 4.3.3.1 there is no constituency at the DSS layer. The embedded **dss** attribute inside both the **bls** and **rls** points to the particular facts inside the top-level, global **dss** feature, that C realizes at the DGS layer. An example of initial revision workspace for the application of a rule to an embedded draft constituent is given in Appendix C. The address parameters of the revision actions inside the example rule given in Fig. 4.17 should be interpreted in the context of a workspace built *locally* at the level of the game result cluster.

How should be implemented in FUF such general revision rules that can be applied locally at various depths inside the draft linguistic structure? The first methods that comes to mind is to use *relative* paths as the address parameters of the revision actions. However, as explained in Section B.1.3.3 of Appendix B, while such paths are perfectly fine in an FG, in an FD they can introduce ambiguity. We have also seen that FUF can unify two FDs or one FD with an FG but *not* two FGs.

Since the search for the revision rule to trigger is implemented as the unification of the draft with an FG encoding the revision rule base, the draft can only be encoded as an FD and not as an FG. Consequently, it cannot contain relative paths.

To get around this difficulty, the reviser *simulates* the local application of general rules by translating the local addresses written in the revision actions into global addresses inside the FD encoding the whole draft. This global FD is the data structure onto which the actions are physically applied. Consider for example, the application of the nominalization revision rule Fig. 4.17 to the game result cluster: *"Orlando defeated Toronto 99-92"*. Since in the DGS of the whole draft sentence, *"O'Neal scored 39 points as Orlando defeated Toronto 99-92."*, this game result cluster appears as a temporal circumstantial, the address **{}** of the initial deletion action (on line 4) of this revision rule is translated as **{circum time}**.

4.3.3.2.3 The monitor: controlling the revision process The revision monitor is the highest level component of the reviser and controls the overall revision process. It works by repeatedly performing the following steps:

1. Take the first element F_1 in the ordered list of floating facts to opportunistically incorporate to the draft.
2. Attempt to incorporate it to the current draft D_0 by calling the revision rule interpreter on the pair $< D_0, F_1 >$.
3. (a) If a revision rule matches this input pair, the interpreter returns a revised draft D_1, expressing the content of both D_0 and F_1: proceed to step 4 below.
   (b) Otherwise the interpreter returns back D_0: start over from step 1 above with the next element F_2 in the ordered list of floating facts; F_1 has been ignored for lack of a revision rule to incorporate it to the draft.
4. Extract the DGS layer from D_1 and pass it to SURGE.
5. Inspect the resulting sentence, measuring both its lexical length and syntactic depth.
6. (a) If the length is below 45 words and the depth below 10 embedding levels, output the sentence: start over from step 1 above with the next element F_2 in the ordered list of floating facts. F_1 has been successfully incorporated to the draft.
   (b) Otherwise output a warning message indicating that the maximum sentence complexity has been reached and stop the revision process; F_1 has been ignored for lack of space.

There are two remarks to be made concerning this simple control mechanism. First, the list of floating facts is actually a list of lists, where each sublist contains a set of *related* floating facts. This is the case because floating facts (especially historical ones) need an anchor point to be included into the draft. For example, the record property of an additional game statistic cannot be incorporated to the report before the incorporation of the statistic itself. These two floating facts will thus be put in the same sublist. Second, because the revision interpreter always picks the first < draft-subconstituent , revision-rule > pair that matches, the monitoring algorithm above may in some cases produce a sentence that is sub-optimal in the sense that it may not contain the maximal number of floating facts that could be fit within the complexity limits. When the threshold is reached, it may be the case that, had another < draft-subconstituent , revision-rule > pair been chosen for some earlier revision, it would have resulted in a more compact form, ultimately allowing the addition of another floating fact without reaching the threshold. The implementation of a backtracking facility within the reviser that would avoid such situations. It however has been left for future work. I come back to this issue in Section 7.2.2.2.

4.4 STREAK at work: two example runs

Having seen how each component of STREAK works separately in the previous three sections, I now comment two example runs showing how they work together to generate a summary. The first example run illustrates how STREAK generates a complex lead sentence, by first producing a simple sentence containing only the obligatory fixed facts and then incrementally incorporating the complementary floating facts through a series of revisions. It also shows how STREAK controls the revision process and decides when to halt it. The second example illustrates how STREAK takes into account the surface form of the current draft to choose which revision operation to use for incorporating a new floating fact into the draft. In this example, STREAK first builds two different draft forms from the same set of input fixed facts. It then incorporates the same input floating fact to the different draft forms, using a different revision rule in each case.

In order to prevent this section from growing over-lengthy, further example runs, illustrating other interesting aspects of the system are given in Appendix C. This appendix contains:

- A set of draft building stages that illustrates the paraphrasing power encoded in the phrase planner and the lexicalizer by producing a variety of draft forms from the same set of input fixed facts.
- A set of parallel revision increments that illustrates the paraphrasing power encoded in the reviser by showing how the same floating fact can be incorporated into the same initial draft form by using different revision rules resulting in a variety of revised draft forms.
- Another full run example that illustrates the locality of the revision rules by repeatedly applying the same rule at a different levels of the current draft structure for different revision increments during the generation of a complex lead sentence.

The fullest account of the two example runs presented in this section can be found in that appendix, where they are repeated, but this time in conjunction with the FDs encoding the complete semantic input and/or draft representation.

4.4.1 Chaining revisions to generate a complex sentence from a simple draft

Example *Run 1*, shown in Fig. 4.19, illustrates three aspects of STREAK:

- How it generates a complex lead sentence, by first producing a simple sentence containing only the obligatory fixed facts and then incrementally incorporating the complementary floating facts through a series of revisions.
- How it controls the revision process and decides when to halt it.
- The variety of revision tools it implements, since a different one is used at each generation increment.

4.4.1.1 Building an initial draft

In Fig. 4.19, the first line contains calls to the top-level functions for the draft and revision passes. As input, the function `draft` takes three arguments:

- The DSS FD containing the input fixed facts to convey in the draft. In the example of Fig. 4.19 this input DSS is built by calling the function `dssF0`. The code of this function is given in Fig. C.3 of Appendix C.
- A partial SSS constraining the form of the output draft by specifying some desired features in the initial draft plan. The phrase planner can only choose options that are compatible with this partial specification of its output. This argument is optional and introduced by the keyword `:sss`. These surface form constraints are generated by calls to functions such as `form-flag1` described below.
- A partial DGS constraining the form of the output draft by specifying some desired features in the initial draft skeletal lexico-syntactic tree. The lexicalizer can only choose options that are compatible with this partial specification of its output. This argument is optional and introduced by the keyword `:dgs`[7].

`draft` returns an three-layer FD encoding the initial draft and as a side effect prints the natural language sentence resulting from the unification of this FD with SURGE.

The function `revise` takes three arguments:

- An three-layer FD encoding the initial draft.
- A list of list of floating facts to attempt to incorporate to the draft. Each sublist contains a set of related floating facts. These sublists are in ordered by decreasing importance of the facts they contain. Floating fact lists of lists are generated by calls to functions such as `float-stack-F`. The code for this function is shown in figures C.4 to C.8 of Appendix C
- The `:verbose flag`, which, when set to T, prints after each revision increment the lexical length (lex-num) and syntactic depth (depth) of the revised draft.

The value of the `(form-flag1)` parameter passed to the `draft` function in Fig. 4.19 is:

```
((rels ((co-occur none)
        (time ((elts ((cdr ((car ((struct hypotax)))))))))))
```

As explained in Section 2.4 an initial draft contains four fixed facts: the main statistic of a player from the winning team, the result of the game (including its final score), its location and its date. Following the corpus observations, STREAK always convey the location as a header. The sentence itself thus contains three main constituents, one for each of the three remaining fixed facts. The form flag above the type of rhetorical relations that the phrase planner can use to group these three constituents. The one above specifies that:

- The draft must be hypotactically structured (indicated by the presence of a `rels` feature at the top-level).
- Two dependent constituents must be grouped in an paratactic structure itself linked to the main constituent by a temporal relation (indicated by the presence of a feature `(time ((elt ...)))` under the `rels` feature).
- The second element in this paratactic structure must be itself hypotactically structured (indicated by the embedded `((struct hypotax))` feature).

When unified with the possible top-level draft structures observed in the corpus and encoded in the phrase planner of STREAK, this specification results in sentences like *Draft 0* at the top of Fig. 4.19. In such

[7] The two optional argument `:sss`, `:dgs` provide a basic facility to systematically test the paraphrasing power of STREAK. The development of more powerful facilities has been left for future work and is discussed in Section 7.2.2.2

```
> (revise (draft (dssF0) :sss (form-flag1)) (float-stack-F) :verbose T)

Draft 0:
Dallas, TX -- Charles Barkley registered 42 points Friday night as the Phoenix Suns
routed the Dallas Mavericks 123 - 97.

Draft 1 (lex-num = 27  depth = 7):
Dallas, TX -- Charles Barkley registered 42 points Friday night as the Phoenix Suns
handed the Dallas Mavericks their 27th defeat in a row at home 123 - 97.

Draft 2 (lex-num = 29  depth = 8):
Dallas, TX -- Charles Barkley registered 42 points Friday night as the Phoenix Suns
handed the Dallas Mavericks their franchise worst 27th defeat in a  row at home
123 - 97.

Draft 3 (lex-num = 34  depth = 8):
Dallas, TX -- Charles Barkley registered 42 points and Danny Ainge added
 21 Friday night as the Phoenix Suns handed the Dallas Mavericks their
franchise worst 27th defeat in a row at home 123 - 97.

Draft 4 (lex-num = 39  depth = 8):
Dallas, TX -- Charles Barkley registered 42 points and Danny Ainge came
off the bench to add 21 Friday night as the Phoenix Suns handed the Dallas
Mavericks their franchise worst 27th defeat in a row at home 123 - 97.

Draft 5 (lex-num = 43  depth = 8):
Dallas, TX -- Charles Barkley matched his season record with 42 points
and Danny Ainge came off the bench to add 21 Friday night as the Phoenix
Suns handed the Dallas Mavericks their franchise worst 27th defeat in a row
at home 123 - 97.

Draft 6 (lex-num = 46  depth = 8):

((SSS ((DSS .... ))))
>
```

Figure 4.19: STREAK generating a simple draft and incrementally revising it into a complex one

sentences, the main clause conveys the main statistic with a list of two dependent constituents as a temporal adjunct. The first element of this list is the nominal conveying the date of the game and the second is the clause conveying the game result.

4.4.1.2 Revising the initial draft

The first sub-element in `float-stack-F` is an historical background fact of type streak extension. It notes that the reported game marks the 27th time that Dallas is defeated on their home turf. To add this fact to *Draft 0*, STREAK uses the non-monotonic `Nominalization` revision rule whose code was given and explained in detail in section 4.3.3.1. This rule is applied to the initial draft game result clause "*the Phoenix Suns routed the Dallas Mavericks*". It replaces the full verb clause pattern "WINNER rout LOSER" conveying the game result in *Draft 0*, by the semantically equivalent support verb clause pattern "WINNER hand LOSER a defeat" in *Draft 1*. Since the game result is now realized by an NP, the expression of its consequence, the updated length of Dallas' losing streak, can be concisely conveyed by adjoining the discontinuous ordinal "*27th ... in a row*" modifying the NP head "*defeat*". The restriction of this streak to home games is conveyed adjoining another modifier, the PP qualifier "*at home*". The variety of nominalization rule used for this revision is thus: `Nominalization with Ordinal and Qualifier Adjoin`.

The second sub-element in `float-stack-F` is another historical background fact, of type record breaking. It brings additional information about the preceding streak extension fact by noting that as a result of this latest extension this streak is now of record length. To add this fact to *Draft 1*, STREAK uses the monotonic revision rule `Adjoin of Classifier to Nominal`. This rule is applied on the very nominal that was created during the preceding nominalization revision ("*their 27th defeat in a row at home*") modifying it with the classifier "*franchise worst*" that expresses its record breaking nature. This illustrates how the choice of a revision rule for a given increment constrain in some cases the range of choices for subsequent increments. This type of `Adjoin` revision rule allows for a very concise expression of the added floating fact. It does not change the syntactic depth of the draft and lengthens it by only two words. After this addition, the draft is only 29 words long, still comfortably below the 45 word limit observed in the corpus.

The third sub-element in `float-stack-F` is a non-historical fact of type additional statistic. To add this fact to *Draft 2*, STREAK uses the monotonic revision rule `Coordinative Conjoin of Clause`. This rule was applied to the main statistic clause, "*Charles Barkley registered 42 points*", because this additional statistic also concerns a player of the winning team. Furthermore, since they are both scoring performances, STREAK exploits this fact and chooses to elide the head of the object in the added conjoined clause (resulting in the phrase "*Danny Ainge added 24 ∅*" instead of "*Danny Ainge added 24 points*"). This illustrates how STREAK opportunistically takes advantage of the particular draft context into which a floating fact is woven, to choose a more concise expression for that fact. STREAK also uses this context to make the most appropriate lexical choice, as illustrated by the choice of the verb "*to add*" for this second statistic. Such a verb can be chosen only in this particular context. It would be inappropriate for example, to realize the main statistic, for which STREAK chose the more general verb "*to register*"' in this particular run. The code for the choice of such verbs was given in Section 4.3.2. How the reviser passes such contextual information to the lexicalizer was also explained in that section.

The fourth sub-element in `float-stack-F` is non-historical fact of type player status. Just as the second floating fact underlined the significance of the first by conveying its record breaking nature, this fourth fact underlines the significance of the third. It notes that the player whose scoring statistic was just added to the draft ("*Danny Ainge added 24*"), is a reserve player[8]. To add this fact to *Draft 3*, STREAK uses the monotonic revision rule `absorb of clause into clause as a result adjunct`. Moreover, it uses the specialization of this revision rule that involves the side transformation `Agent Control`. This specialization is chosen when STREAK notices that both the absorbed and absorbing clauses share the same agent. It allows the agent of the absorbed one, which was part of the original draft, to be deleted, resulting in "*Danny Ainge came off the bench ∅ to add 24*" instead of "*Danny Ainge came off the bench for Danny Ainge to add 24*", thus opportunistically gaining space and fluency. This example illustrates the capability of STREAK to use idioms such as the expression "*to come off the bench*" which conveys that a player is a reserve.

[8]Making the fact that he scored that many point all the more remarkable.

The fifth sub-element in `float-stack-F` is a historical fact of type record equalling. It concerns the main statistic. To add this fact to *Draft 4*, STREAK uses the revision rule `adjunctization of created into instrument`. It moves the object of the main statistic clause that was filling the `created` role in that clause[9], to an `instrument` role in order to accommodate the added record as object. The equalling aspect of this record is expressed as the new main verb "to match" replacing the original verb "*to register*". The action explicitly conveyed by this original verb is now implicitly conveyed by "*to match*", since matching a record can only come as a consequence of a performance. This example thus illustrates the ability of STREAK to opportunistically take advantage of the addition of a new fact to gain space by making part of the realization of another fact already in the draft implicit. It also demonstrates how STREAK takes into account stylistic conventions observed in the corpus. Compare the addition of this fifth floating fact with the addition of the second one. They both concern a record, the difference between them being that the second fact expresses that a record was *broken* and the fifth one that it was merely *equalled*. This difference, which could seem minor at first, triggers the use of entirely different revision tools: the monotonic `Adjoin` for the second fact and the non-monotonic `Adjunctization` for the fifth one. This difference in strategies implements the stylistic convention observed among sports or stock market writers that mentioning of a record update event without explicitly specifying whether it is of the breaking or equalling type implies that it is of the breaking type. This convention allows STREAK to use the simple and concise revision tool `Adjoin` for record breaking events: note how nothing in *Draft 2* specifies whether the 27th defeat of Dallas actually breaks or merely equals their longest losing streak. Using such an implicit form for record equalling events as well would be misguiding however. The need to keep reports concise must be balanced with the need to keep these two type of events unambiguously distinguishable. It is in order to be explicit about the equalling type of the record update event added in the fifth increment, that STREAK uses the less concise and more complex `Adjunctization` revision.

After the addition of this fifth floating fact, the draft is only two words away from the maximum length of 45 observed in the corpus. Thus, unless the next sub-element in `float-stack-F` can be added with only two more words, it will not fit in this lead sentence summary. This next sub-element is an additional statistic, the passing performance of Danny Ainge. The most concise way it can be accommodated in the draft is by revising the nominal realizing the scoring statistic of this player that was added during the third revision increment and which is already reduced to the cardinal number "*21*". STREAK applies the revision rule `Coordinative Conjoin of Nominal` to this nominal, yielding "*to add 21* **and 7 assists**". This revision thus adds three new words (in bold) while not deleting any and thus pushes the revised draft over the length limit. STREAK thus halts the revision process without printing the draft resulting from this final revision. As final value, the `revise` function returns the three-layer FD representing the previous draft, which was under the complexity limits. Since this FD is very large, its body is not shown in Fig. 4.19, but its presence is signaled by the abbreviation `((SSS ((DSS ...  ))))`. The full detailed body for this final draft representation is given in Appendix C, however.

4.4.2 Choice of revision rule constrained by the surface form of the draft

Example *Run 2*, shown in Fig. 4.20, illustrates how in STREAK, the choice of a revision rule to add a given floating fact onto the draft is sensitive not only to the content of the draft, but to its surface form as well. This run starts with two calls to the function `draft` to build two alternative draft forms, from the same input but a with different realization constraint. The common input, is a DSS FD encoding the four fixed facts to convey in both draft. It is built by calling the function `dssC0`. For the first call to `draft`, the realization constraint is `(form-flag1)` whose value was given in the previous section. It constrains the output `draftC1` to express the game result as a *full* verb clause that follows the pattern "WINNER full-verb LOSER" and is subordinated to the main statistic clause as a time adjunct. In this particular run, the full verb chosen (randomly) by the lexicalizer is "*to beat*".

For the second call the realization constraint is `(form-flag2)` whose value is

```
((rels ((co-occur none)
```

[9] cf. sections B.2.2.1 and B.3.2 of Appendix B for the definition of the thematic roles used in STREAK.

```
(time ((elts ((cdr ((car ((struct event)))))))))))
```

It constrains the output `draftC2` to express the game result this time as a *support* verb clause following the pattern "WINNER support-verb LOSER nominal"[10]. In this particular run the support-verb/nominal-head collocation chosen (randomly) by the lexicalizer is "*to nail down a win*".

Once these two alternative synonymous draft sentences have been built, the function `revise1` is then called on each of them with the same additional floating fact `adssC4` as second parameter. `revise1` is the function to call for a single revision increment. It implements the revision rule interpreter described in Section 4.3.3.2. In contrast, the function `revise` called for example *Run* is used for chaining revisions and implements the revision monitor described in Section 4.3.3.2.3. `revise` works by traversing the list of lists of floating facts and repeatedly calling `revise1` on each fact. `adssC4` encodes a losing streak extension for the Boston Celtics. To incorporate this floating fact to `draftC0`, STREAK uses the revision rule `Nominalization`. In contrast, to incorporate this same floating fact on the synonymous but linguistically distinct `draftC1`, STREAK uses the revision rule `Adjunctization`. In each case, the choice of one revision rule over the other is motivated by the surface form of the respective drafts involved. `Nominalization` realizes the new fact by modifiers attached to a nominal resulting from the transformation of a full-verb clause into a support-verb clause. `Adjunctization` conversely replaces a support-verb clause by a full-verb clause incorporating the new fact by a full-verb and a new object while displacing the original object to an adjunct position. Since `draftC1` follows a full-verb pattern, only `Nominalization` and not `Adjunctization` is applicable to it. For `draftC2` following a support verb pattern, it is just the opposite. There is no game result NP in `draftC1` to be adjunctized and no game result full verb in `draftC2` to be nominalized. It is precisely because the applicability of revision rules such as the two above is dependent on surface form, the presence of the DGS layer in the draft representation of STREAK is required.

4.5 STREAK by the numbers

STREAK was implemented on top of an extended FUF/SURGE package. In this extended package, FUF-5.3 consists of 552K of COMMON-LISP code, including 36K of entirely new code[11] for the non-monotonic manipulation of Functional Descriptions (FDs), and SURGE-2.0 consists of 264K of FUF code, including 35K of entirely new code for the extended system for adverbial clause elements.

These two extensions to the FUF/SURGE package were essentially preparatory work, paving the way for the core implementation of STREAK. This core consisted of implementing the revision rule interpreter and encoding the linguistic data compiled during the corpus analysis as three declarative knowledge sources:

- The phrase planning rule base.
- The lexicalization rule base.
- The revision rule base.

The revision rule interpreter consists of 37K of both COMMON-LISP and FUF code. Each rule base is encoded as a Functional Grammar (FG). As explained in Appendix B, an FG is a disjunction of conjunctions of features, with each feature potentially a recursive disjunction of conjunctions of sub-features. An FG can thus be viewed as an and/or-tree of options and the best way to quantify the number of cases covered in an FG is to count the number of disjunction-free conjunctions that it contains. Each such conjunction is only one level above the bottom of the tree and corresponds to a disjunction-free rule. The FG for phrase planning encodes about 130 such rules, the one for lexicalization encodes about 29,043,800 such rules (covering the various senses and thematic usages of 115 open-class words[12] in the domain sublanguage) and the one for revision encodes about 1,510 such rules.

[10] Like `(form-flag1)`, `(form-flag2)` also constrains the game result clause to be subordinated to the main statistic clause as a time adjunct.

[11] Written by Elhadad.

[12] These words include neither proper nouns and quantitative values which are passed from the input to the generator, nor function words which get added by SURGE.

```
> (setf draftC1 (draft (dssC0) :sss (form-flag1)))  ;; Draft form 1

Hartford, CT -- Karl Malone hit for 39 points Friday night as the Utah
Jazz beat the Boston Celtics 98 - 94.

((SSS ((DSS .... ))))
> (setf draftC2 (draft (dssC0) :sss (form-flag2)))  ;; Draft form 2

Hartford, CT -- Karl Malone notched 39 points Friday night while the
Utah Jazz nailed down a 98 - 94 win against the Boston Celtics.

((SSS ((DSS .... ))))
> (revise1 draftC1 (adssC4))  ;; Revising Draft 1 using Nominalization

Hartford, CT -- Karl Malone hit for 39 points Friday night as the
Utah Jazz brought the Boston Celtics their sixth consecutive setback at home
 98 - 94.

((SSS ((DSS .... ))))
> (revise1 draftC2 (adssC4))  ;; Revising Draft 2 using Adjunctization

Hartford, CT -- Karl Malone notched 39 points Friday night while the Utah
Jazz extended the Celtics' homecourt losing streak to six with a 98 - 94 win
 against Boston.

((SSS ((DSS .... ))))
>
```

Figure 4.20: STREAK using different revision rules depending on the surface form of the draft

These three FGs respectively occupy 36K of FUF code for the phrase planning rule base, 37K of FUF code for the lexicalization rule base and 56K of FUF code for the revision rule base. The fact that the lexicalization FG encodes several orders of magnitude more rules than the phrase planning FG while being essentially of the same size illustrates the high level of code compactness achievable by relying on the the run-time compositional instantiation (through recursive functional unification) of rules whose common features are shared inside FG fragments.

Chapter 5

Evaluation

In this chapter, I *quantitatively* evaluate several aspects of the research presented in this dissertation. Different aspects are evaluated along different dimensions using different test corpora. After surveying the general problematic of evaluation in the context of language generation, I first describe the use of two new basketball report corpora to evaluate:

- The coverage of the corpus analysis results (ontology, realization patterns and revision tools) presented in Chapter 2.
- The robustness of the corpus analysis methodology (presented in the same chapter) as a knowledge acquisition approach.
- The gains in terms of both coverage and robustness of the revision-based approach to generation presented in Section 3.3 over the classic one-shot approach.

I then describe the use of a stock market report corpus to evaluate the cross-domain portability of the revision tools presented in Section 2.5 of Chapter 2.

5.1 Evaluation and language generation

5.1.1 The difficulty of the issue

Although a potentially vast field in its own right, evaluation remains to date an almost completely untouched area of generation research. Among the landmark dissertations centered around the development of a generation system, only one contains an entire chapter specifically dedicated to evaluation: Kukich's [Kukich 1983]. Her system ANA and Elhadad's ADVISOR II [Elhadad 1993b] are the only two generation systems that have been the object of *quantitative* evaluation. These evaluations are described and contrasted with the evaluation work presented in this thesis in Section 6.4.

The paucity of evaluation efforts in generation is rooted in the extremely challenging nature of the task. There are multiple reasons for this difficulty, notably:

- The variety in input representations, target outputs and application domains of existing generation systems, limiting the potential for comparative evaluations.
- The high subjectivity of what constitutes a "good text", limiting the potential for absolute qualitative evaluations.
- The frequent unavailability of large, systematic Input/Output test sets, limiting the potential for quantitative evaluations.

In the following subsections I review each of these difficulties in turn. I then explain how some of them can be alleviated when textual corpora are available.

The subjectivity of "text quality"

Evaluating writing - even human writing - has always been a thorny issue. While humans routinely make judgments about the quality of the prose they read, these judgments are highly subjective. Even such basic evaluation attempts as assigning a grade level to a text has been the object of endless controversies. This is due to the fact that judgments of written prose rely on a vast array of implicit and goal-dependent criterion and draw upon many vague and intuitive notions. Making these goals and criterion explicit and rigorously defining these notions is an intriguing but daunting task. It is, however, a pre-requisite for the absolute evaluation of the "quality" of texts (whether generated by computers or humans) in a systematic and objective way.

The variety of input representations and target output texts

Comparative evaluations are just as problematic as absolute ones[1].

The first difficulty for comparing the respective merits of different generation systems is the nature of their input. With a few exceptions (*e.g.,* stand-alone portable syntactic components, generation sides of transfer-based translation systems, report generators working all the way from number tables), the input to a generator consists of a *semantic representation* of the content to convey, sometimes annotated with communicative intent. No broadly accepted standard representation scheme for semantic content and intentions has yet emerged from knowledge representation research. Comparing generators working from drastically different input representations is therefore a murky business.

The second difficulty is that, until now[2] no two generators have been conceived and implemented to produce the same set of target outputs. Existing systems have been developed for different domains and in the rare cases when several generators exist in the same domain, they use a different language (*e.g.,* ANA [Kukich 1983] in English and FRANA [Contant 1986] in French for the stock market domain). Different domains do not only mean different encyclopedic contexts but also different writing styles. Whereas a telegraphic style is well-suited for a weather forecast it would be completely inappropriate for a business letter. Similarly, whereas a fact-packed and highly metaphorical style is best for newswire sports stories it could not be used for technical documentation. Indeed, it is interesting to note that the newswire summaries of the corpora I analyzed did not follow the stylistic heuristics that writing experts recommand for improving the clarity of scientific papers. For example, support verbs were pervasive in the corpus and the subject head noun was often separated from the verb by lengthy relative clauses or appositions. These two expressive forms are strongly objected by [Gopen and Swan 1990] for scientific writing. This discrepancy in style can be explained by a discrepancy in goals. In a science article the main goal is to introduce the reader to new, complex concepts. In a newswire story it is rather to pack information concerning to familiar, simpler concepts. Since in the latter case the reader is not burdened with the complexity of the content, she can handle the more complex prose which allows fitting many facts in a short space.

The unavailability of systematic I/0 test sets

Not only do generators lack a reference standard for input and output, but systematic test sets of either are not always available.

Due to the overall complexity of the generation task, many recent research efforts, like the one presented here, have focused on restricted sub-tasks in order to attack them in depth. These efforts thus resulted in the

[1] At least those that concern particular *systems* and are carried out directly on *output texts*. In Section 5.2 I present a comparative evaluation of two generation *models* carried out on the different *knowledge structures* which are used by each model.

[2] To the best of my knowledge.

implementation of generator *components* as opposed to complete text generation systems. In such situation, some of the implemented components will inevitably work from an input representation that comes, in the overall generation architecture, from other components whose task is out of the research scope at hand and thus unimplemented. Evaluating such components can only be performed by using hand-coded (and thus necessarily small) input test sets.

In some applications, a systematic test set is not available not only for the input to the generator, but for its target output as well. While some generation applications aim at saving the cost of using humans experts with other pressing responsibilities from writing texts, other applications aim at providing documentation or summaries that are needed yet currently missing for lack of available writing manpower. In the latter case, there is no corpus of human-written target texts available onto which to test the output of the generator. The situation is similar for on-line help systems: the very need for a generator responding dynamically to a specific dialog situation instead of simply displaying pre-stored text precludes the use of manual pages as model texts. For such applications, a set of target responses have to be initially hand-coded and then incrementally adjusted through user feedback and determining the quality of the generated response requires determining how well it meet the user needs. Proposed evaluations [Hirschman and Cuomo 1994] often center around time to task completion, where the user is given a task that requires use of the system to solve. Such evaluations are problematic because the ease with which a user can request information also affects the result and thus do not allow assessing the respective effects of the understanding and generation components.

The opportunity created by corpus data

For generation applications where human generated text corpora are available as model output sets, setting up an evaluation is much easier. One reason why ANA [Kukich 1983] could be quantitively evaluated was that both its input (number tables) and target output (newspaper articles) were available in a systematic way.

Corpus data allows better circumscription of the goal of a generation system: it is to produce texts that match as closely as possible the texts from the corpus. It allows viewing the writing process as a black box. Such a view focuses on the *objective* task of *observing what* human writers generate in a given situation and avoids the more *subjective* task of *speculating why* they do so. The quality of a text produced by a generation system can then be defined in terms of how distinguishable it is from a text that a professional writer would produce from the same input data. When the corpus is large enough, such an evaluation can become quantitative. When model texts from multiple writers are available, stylistic idiosyncrasies can be filtered out by encoding in the generator only the most commonly used expressive forms. A corpus-based approach shifts the task of the evaluation from defining what constitutes a “good text” to assessing the representativity of the textual corpus used as model. For this latter task, developing automatic or at least semi-automatic methods is much more feasible.

5.1.2 The rising interest in the issue

While evaluation has been largely ignored by the generation community up to now, it is bound to become a central issue fairly quickly. Two different sets of forces are at play in this emergence. The first set is internal to the field. The second is external. I briefly discuss each in turn in the following paragraphs.

Maturity of the field

Natural language generation seems to have reached a turning point in its maturation. After the initial exploratory phase when most research consisted in discovering new problems and providing initial solutions to them, more efforts are now being put into taking a second and more focused look at well-known problems to improve on the initial solution. Since the contribution of such research does not lie in the originality of the problem itself, new approaches must be compared convincingly and advantageously with previous ones. This is encouraging the development of comparative evaluations. It is also promoting quantitative evaluations. Even though their real significance may at times be questionable, evaluations that can be summed up by a few numbers tend to convince broader audiences than qualitative evaluations which require deep understanding of subtle details.

Influence of related fields

The vogue of comparative and quantitative evaluations is also spreading from their popularity in fields close to generation, namely language understanding and statistical NLP. Events entirely dedicated to quantitative evaluations such as the Message Understanding Conference [MUC-4 1992] have now been held for several years in natural language understanding, a field that is older than generation and beneficiates from a larger workforce. Evaluation has been a central issue from the start in statistical NLP[3] a line of work aiming at developing tools that do not rely on any hand-coded knowledge but are instead trained using weak learning heuristics on very large textual corpora. This early emphasis on evaluation in statistical NLP is rooted in the two fundamental differences that sets it NLP apart from its knowledge-based counterpart. First, statistical NLP is experimental in nature while knowledge-based NLP is analytical. This experimental flavor where simple approximations are quickly tried out on data allows to invest more time in evaluation than when exact knowledge is painstakingly abstracted from in-depth data analysis. Second, statistical NLP seeks to attain wide coverage at the expense of accuracy, while knowledge-based NLP opts for the other side of this trade-off. Inaccuracy being more immediately visible than brittleness, the issue of evaluation became crucial more rapidly.

Because of these fundamental differences in goals between these two fields, evaluation techniques that originated in statistical NLP cannot be used "as is" for knowledge-based NLP. Both the object and the method of evaluation need to be carefully adapted. In terms of what should get evaluated, efforts are best focused on the weak point of each field. Consequently while in statistical NLP evaluation schemes should emphasize accuracy over robustness, in knowledge-base NLP evaluation schemes should do just the opposite. In terms of evaluation methods, although textual corpora can be used in both cases, each field use these corpora in a different way. In statistical NLP, corpora are input data from which to automatically extract shallow approximations of deep knowledge. Consequently, corpora need to be very large in order to determine whether the correlation between the proposed shallow approximation of the deep knowledge and the deep knowledge itself is statistically significant. In knowledge-base NLP, corpora are analysis material from which to manually extract the deep knowledge itself. Corpora size requirements are thus greatly reduced, since with the help of human expertise, deep knowledge can be acquired from textual corpora of modest size.

5.1.3 The diversity of the issue

Having surveyed both the difficulty and the growing importance of the issue of evaluation in language generation I now turn to its diversity. I identify various dimensions along which different types of evaluations can be characterized. Some of these dimensions concern the particular *object* of the evaluation while others concerns the *methods* used for the evaluation. I review each in turn in the following paragraphs.

What to evaluate?

Evaluating a generation system can involve evaluating either:

- The issues that it addresses or the solution that it provides.
- The underlying model on which it is based or its particular implementation.
- Its robustness or the accuracy of its output.

[3] It is important to keep in mind that the recent revival of the use of statistics in place of deeper knowledge in NLP research has focused on applications such as translation, summarization and information retrieval. The rationale is not to use statistics to *perform* either understanding or generation, but to hope that for such applications, both tasks can be somehow *bypassed*.

- How well and/or robustly does it performs the various subtasks of language generation (content determination, content organization, content realization).

When evaluating a research prototype (whether in language generation or not) one must distinguish between evaluating the importance of the specific *issues* addressed by the system and evaluating the *solutions* that the system brings to these issues. For example the percentages of corpus sentences with floating concepts and/or historical information presented in Chapter 2 evaluates the importance of these two issues. These percentages do not, however, evaluates the solution that STREAK offers to these issues.

In evaluating a solution, it is also important to distinguish between evaluating the underlying generation *model* of the system and its particular *implementation*. For example in [Kukich 1983], Kukich quantitatively evaluates the coverage of the generator ANA, which is a particular implementation of the general one-pass macrocoded generation model that she proposes. The model in itself is not quantitatively evaluated in its abstract generality.

A generator can be evaluated directly in terms of its *output* or more abstractly in terms of the *knowledge structures* that it relies on to produce this output. It is also quite a different issue to evaluate the *robustness* of a generator and to evaluate the *accuracy* of the texts that it produces. These last two contrasts are related. For example, a generator using canned text can produce texts that perfectly mimic the corresponding texts generated by human writers for a small sample of input data. However, such a system will break down when presented with input data outside of that small sample. In contrast, a generator relying on highly abstract and compositional knowledge structures will be robust enough to produce satisfactory texts from input data outside the initial set of data that was used to acquire these structures.

Robustness can be decomposed in different facets:

- *Coverage*: how much of a given domain's total sublanguage is covered by the encoded knowledge structures.
- *Extensibility*: how many more such knowledge structures would be needed to cover the whole sublanguage.
- *Portability*: how domain specific are those knowledge structures.

Similarly accuracy can also be decomposed in different facets:

- *Syntactic*: dealing with the grammaticality of the generated sentences.
- *Semantic*: dealing with the mapping from conceptual content to linguistic form.
- *Lexical*: dealing with constraints among words such as collocations.
- *Discursive*: dealing with factors such as coherence and high level textual organization.
- *Stylistic*: dealing with factors such as conciseness and readability.
- *Interpersonal*: dealing with factors related to the intended reader.

Inaccuracies of the first three types above (and even more strongly for first two) tend to be so blatant that they are rarely even discussed in the research literature. They are in general seen more as "bugs" than as inaccuracies. Consequently, only a polished version of a generator that is essentially free of such inaccuracies is considered a complete implementation. Most of the discussion on accuracy focuses on the last three facets.

How to evaluate?

Having reviewed the various aspects of a generator that can be evaluated, I now survey the different methods that can be used for each of these aspects. Evaluation methods can be either:

- Qualitative or quantitative.
- Absolute or comparative.

- Based on human judgments or on corpus data.
- Manual or automatic.

An example of qualitative *vs.* quantitative evaluations in the context of language generation can be found in [Kukich 1983], p.137-138. A *qualitative* evaluation of the syntactic coverage of ANA is first given, listing all syntactic forms observed in a corpus of stock market reports and specifying which forms are encoded in the generator (*e.g.,* participial clauses) and which are not (*e.g.,* infinitive clauses). It is followed by a *quantitative* evaluation of syntactic coverage counting the proportion of corpus sentences containing only constructs encoded in ANA.

These two evaluations are *absolute*, concerning only ANA. In Section 5.2 of the present thesis, I present a *comparative* evaluation which contrasts the respective robustness of the two-pass microcoded generation model on which STREAK is based with the one-pass macrocoded generation model on which previous report generators such as ANA [Kukich 1983] are based.

All the evaluations mentioned until now are based on corpus data. One could also imagine evaluating a generator by having a human expert (*e.g.,* a sports writer in the case of STREAK) looking at a test set of generated texts and counting the proportion of sentences that are accurate and complete (and optimally concise in the case of a summarization system). This type of evaluation relying on one or several human judges is commonly used in statistical NLP. For example, it has been used to evaluate systems which compile collocations [Smadja 1991] or form groups of semantically related adjectives [Hatzivassiloglou and McKeown 1993]. However, in the context of language generation this type of approach would not yield very interesting results. This is due to the fact that using canned text, output accuracy can be trivially achieved at the expense of robustness, which is the central issue for a generator as it is for most knowledge based systems. In addition, such methods cannot be automated. Given the restricted accessibility of human experts, purely manual evaluation methods tend to be impractical since they cannot be repeated on a regular basis to measure the impact of gradual changes made during the life cycle of the system. Semi-automatic, corpus-based evaluation schemes seem thus preferable.

5.1.4 The evaluation approach of this thesis

Having surveyed the dimensions along which evaluation can vary, I now situate along these dimensions the particular evaluations that I carried out for the research presented in this thesis.

There are two objects of evaluation:

- The new two-pass microcoded language generation model presented in Chapter 3.
- The hierarchy of revision rules presented in Appendix A and resulting from the corpus analysis presented in Chapter 2.

I thus evaluate the new type of deep *knowledge structures* required by this model as opposed to the output of the prototype system which relies on these structures to generate summaries. I also evaluate the generation *model* proposed in this thesis as opposed to the particular *implementation* of this model in the STREAK prototype. I carry out three separate evaluations. Each one measures a different aspect of *robustness*: coverage and extensibility of the revision-based generation model and portability of the revision rules.

The fact that the evaluation concerns the *knowledge structures* needed to generate the texts of the output texts themselves is crucial, since, as noted above, a generator using canned text can produce texts that perfectly mimic the corresponding texts generated by human writers for a small sample of input data. However, such a system will break down when presented with input data outside of that small sample. In contrast, a generator which relies on abstract, compositional knowledge structures should be robust enough to produce satisfactory texts from input data outside the initial set of data that was used to acquire these structures. The question is, how robust?

The second key feature of this evaluation is that it concerns a general *model* and not a particular *implementation*. It is this feature that sets this evaluation apart from previous evaluation work and make its results of considerably wider relevance. As noted in section 6.4, [Kukich 1983] evaluates the coverage, of the system ANA, a particular implementation of the one-pass macrocoded generation model that she proposes. While necessary for the development of a real-world system, the results of this type of evaluation primarily depend on the amount of effort dedicated to hand-coding the particular knowledge structure needed by the implementation (in the case of ANA, a lexicon of stock market phrases). But since this effort would have to be duplicated for each new domain, such results provide little insights on the general applicability of the underlying generation methodology. Such insights are gained only by evaluations performed at the model level, independently of a particular implementation.

I what follows, I evaluate STREAK's underlying *model* in a way that is:

- Quantitative.
- Based on corpus data.
- Both absolute and comparative.
- Semi-automatic.

In the following sections, I describe an adaptation of the traditional training/test corpus scheme for *quantitative* evaluation to the corpus analysis for generation knowledge acquisition presented in Chapter 2. Considering the initial basketball report corpus on which this analysis was performed as "the training corpus", I analyze two other basketball report corpora and a stock market report corpus as "the test corpora". Since the initial corpus did not literally serve as input data for automatically "training" a system, but instead as analysis data for manually acquiring the system's knowledge, it is more appropriate to refer to it as the "acquisition" corpus than as the "training" corpus.

Using this approach, I present three distinct evaluation efforts. The first defines a set of parameters assessing the limit in *coverage* of the whole target sublanguage of STREAK's application domain, that can be attained by analyzing a one year sample of the sublanguage. This first evaluation is *comparative*. Different parameters are used for measuring the impact, on such coverage limit, of relying on the knowledge structures respectively needed by a two-pass microcoded generator such as STREAK and a one-pass macrocoded generator such as ANA or SEMTEX.[4].

The second evaluation defines a set of similar parameters, but this time measuring the *extensibility* of the respective approaches. It is also *comparative*. The coverage parameters answer the question: with the knowledge structure acquired by analyzing a year sample of the sublanguage, how many sentences from a different year sample can a system which relies on these knowledge structures generate? In contrast, the extensibility parameters answer the question: how many new knowledge structures would the generator need in order to also fully cover a different year sample?

The third evaluation does not directly concern the new generation model proposed in this thesis but instead the new type of linguistic knowledge that is needed for implementing the model: revision rules to incorporate additional content in simple draft sentences. This last evaluation is *absolute*. It estimates the proportion - among the revision rules acquired for the initial sports application domain - that are usable in another domain, finance. This third evaluation measures the *portability* of those rules.

Obtaining the evaluation parameters for all three evaluations required repeating - for each test corpora - most of the corpus analysis steps performed on the acquisition corpus. This task was *partially automated* by approximating the source and target realization patterns of each revision rule by a regular expression of words and parts-of-speech tags. All the test corpus sentences matching a given expression were then automatically retrieved a software tool called CREP (cf. Appendix D and [Duford 1993]) specifically designed for this purpose. Filtering out the incorrect matches resulting from imperfect approximations was then done by manual post-edition.

While I do not quantitatively evaluate STREAK's *implementation* directly, several factors in its design nonetheless contribute to the high accuracy of the text it generates:

[4] The result of this evaluation thus do not directly apply to the one-pass yet microcoded generators based on the Meaning-Text Theory.

- The fact that STREAK uses SURGE as a syntactic front end guarantees that it is *syntactically accurate*. SURGE comes with a large set of test inputs to be run after each change to the grammar. This input test set systematically probes each branch of the grammar and many combinations of features from different branches. The initial input set was incrementally created over seven years during the development of the generation systems COMET [McKeown *et al.* 1990] [McKeown *et al.* 1993], COOK [Smadja and McKeown 1991] and ADVISOR-II [Elhadad 1993b]. The extension of the input set which tests the extensions from from SURGE-1.0 to SURGE-2.0 are given in Section B.5 of Appendix B.
- The fact that STREAK relies only on corpus-observed phrase planning rules, lexicalization rules and revision rules insures that it is essentially *semantically and lexically accurate*. Each individual syntactic construct and vocabulary item that STREAK can use to express each domain concept has been empirically observed in several corpus sentences for the expression of that very concept[5].
- The fact that STREAK monitors corpus-observed limits on the total number of facts (12) words (46) and syntactic embeddings (10) in a single sentence insures that it is fairly *stylistically accurate*.

This enforcement of stylistic accuracy could be improved by observing in the corpus and monitoring in the system finer grained complexity limits *within* sentence subconstituents. In addition, more testing work would be required in order to *fully guarantee* STREAK's semantic and lexical accuracy. This future work is discussed in Section 7.2.4.

5.2 Quantitative evaluation of robustness

5.2.1 Review of acquisition corpus results

In Chapter 2 I presented the analysis of one year of basketball summaries from the UPI newswires. The fine-grained part of the analysis was focused on the lead sentences conveying information from only the four most common semantic classes:

1. Final result and score of the game
2. Streak extending (or interrupting) nature of the result
3. End of the game statistics (for players or teams)
4. Record breaking (or equalling) nature of these statistics

These sentences were made of two top-level syntactic constituents: one grouping facts of classes (1) and (2) - called the game-result cluster - and another grouping facts of classes (3) and (4) - called the statistics cluster.

The fine-grained analysis consisted of identifying the knowledge structures needed for the development of a system generating such sentences. Among these structures,

three types would be needed by any generator, regardless of its architecture:

- Semantic classes of content units (*e.g.,* winning streak extension).
- Combinations of such classes with tokens co-occurring in a corpus cluster (*e.g.,* <game result , winning streak extension>).
- Realization patterns for these combinations (cf. Fig. D.2 p. 345 in Appendix D for an example pattern for the <game result , winning streak extension> combination).

The fourth type, *revision tools* to produce complex realization patterns from basic ones (*e.g.,* Adjunctization of Range into Instrument to add a winning streak to a game result), is specific to the draft and revision generation architecture proposed in this thesis.

[5] As explained in Section 7.2.4, the possibility however remains that in some untested cases, STREAK may produce unfelicitous *compositions* of those individually accurate constructs.

5.2.2 Evaluation goals and test corpora

The evaluation had four main goals:

1. Estimate how much of the whole sublanguage in a given domain[6] is captured by a one-year analysis (coverage).
2. Estimate how many more knowledge structures need to be abstracted to keep up[7] with the new textual data available each year (extensibility).
3. Estimate how much is gained in both coverage and extensibility by using revision tools instead of realization patterns as the knowledge structures on which to base the implementation.
4. Estimate how fast iteratively larger newswire samples converge towards the whole sublanguage.

The first test corpus used for these estimates consisted of lead sentences from the 91-92 season game summaries. These sentences satisfied the same semantic restrictions as for the 90-91 season acquisition corpus (no historical facts other than records and streaks, no non-historical facts other than end of the game statistics) plus a new one: that they contain at least one historical fact. This further restriction was added to reduce the test corpora to manageable size while preserving the most original topic of the thesis - the expression of historical information - in the scope of the evaluation. The semantic filtering of these sentences was done manually while compiling the reports from the news reader. Phrasal patterns and revision rules which were observed in the acquisition corpus (*i.e.,* the 90-91 season) *only* in sentences with no historical facts were excluded from the scope of this evaluation. This insured that the semantic additional restriction above did not bias the results.

In order to test whether the sublanguage captured by analyzing successive reporting seasons quickly converges toward the whole domain sublanguage, I repeated the evaluation using similarly semantically restricted lead sentences from a third season: 92-93. For this second round of evaluation, the acquired corpus consisted of sentences from the first *two* seasons and the test corpus of sentences from the third season.

5.2.3 Evaluation parameters

In this section, I define the evaluation parameters estimating the coverage and extensibility of the knowledge structures abstracted from the acquisition corpus. For most parameters a different definition is needed for, on the one hand, the traditional one-shot generation model (where sentences are produced all at once from the whole conceptual representation of their content), and on the other hand, the new revision-based generation model proposed in this thesis (where optional content is added incrementally to an initial simple draft conveying only obligatory content).

5.2.3.1 Coverage parameters

I first distinguish between four types of coverage:

- *Conceptual* coverage with respect to individual concepts only. It measures the proportion of test corpus sentences not generable with the knowledge structures from the acquisition corpus due to the presence of new[8] *concepts*.
- *Clustering* coverage with respect to concept combinations only. It measures the proportion of test corpus sentences not generable with the knowledge structures from the acquisition corpus due to the presence of a new *combination* of known concepts.

[6] In the example at hand basketball.
[7] By "keep up" here I mean providing full coverage of the observed data.
[8] In the context of this evaluation "new" is opposed to "known" and means not already observed in the acquisition corpus.

- *Paraphrasing* coverage with respect to realization patterns only. It measures the proportion of test corpus sentences not generable with the knowledge structures from the acquisition corpus due to the presence of a new *linguistic forms* for expressing known concept combinations.
- *Realization* coverage with respect to all knowledge structures. It measures the *total* proportion of test corpus sentences not generable with the knowledge structures from the acquisition corpus (whatever the cause).

Recall from Section 2.4 that an important finding of the acquisition corpus analysis was that the statistic and result clusters were independent, *i.e.,*, neither the content nor the form of one influences those of the other. This observation allows us to compute each coverage parameter separately for the statistic cluster and the result cluster and then derive the coverage for whole sentences as the product of the two.

5.2.3.1.1 Conceptual coverage With respect to individual concepts, there are only two possible situations for a test corpus sentence: (1) it conveys an instance of a new concept, or (2) it contains only instances of known concepts. As shown in fig. 5.1, the test corpus T can thus be partitioned in two: C_u comprising the sentences in situation (1) and C_k comprising sentences in situation (2). Conceptual coverage V_c is then simply defined as the size of C_k divided by the size of the test corpus.

Definition 1 Conceptual coverage

$$V_c = \frac{cardC_k}{cardT}$$

5.2.3.1.2 Clustering coverage With respect to concept clustering, the situation is a little more complex. This is due to the fact that a concept cluster can be new for two different reasons: (1) because it includes a new concept, or (2) because it groups concepts which, though individually known, had never been seen clustered together before. Conceptual coverage already accounts for the first case. To keep clustering coverage independent from conceptual coverage, sentences conveying instances of new concepts must be excluded from the computation of clustering coverage. As shown in fig. 5.1, the remaining part of the test corpus, C_k, can then be partitioned in two: G_k comprising the sentences clustering concepts into known concept clusters and G_u comprising the sentences clustering known concepts in a novel way. For one-shot generation, clustering coverage V_g^1 can then be defined as the size of G_k divided by the size of C_k.

Definition 2 Clustering coverage for one-shot generation

$$V_g^1 = \frac{cardG_k}{cardC_k}$$

This simple definition is valid for one-shot generation because in that framework the choice of what concepts to combine in a given sentence is separate from the choice of linguistic form for a given combination. All sentences - whether simple or complex - are built using *concept combination rules* which are independent of linguistic form considerations. Computing clustering coverage therefore does not involve examining realization patterns capturing the alternative linguistic forms.

This is not the case for revision-based generation where complex sentences are built in part using *revision rules* which incorporate both concept combination and linguistic realization knowledge. During revision, new concept combinations and new realization patterns are simultaneously created. What becomes relevant with respect to a concept combination is thus no longer whether it is new or known, but rather whether is it derivable from a known basic combination via known revision rules.

If a concept combination was seen in the acquisition corpus, then revision rules to derive it from a basic combination have been abstracted. Therefore, all known concept combinations are derivable using known basic combinations and known revision rules. However, some unknown concept combination may happen to be also derivable from known basic combinations via known revision rules. Consider for example the following situation:

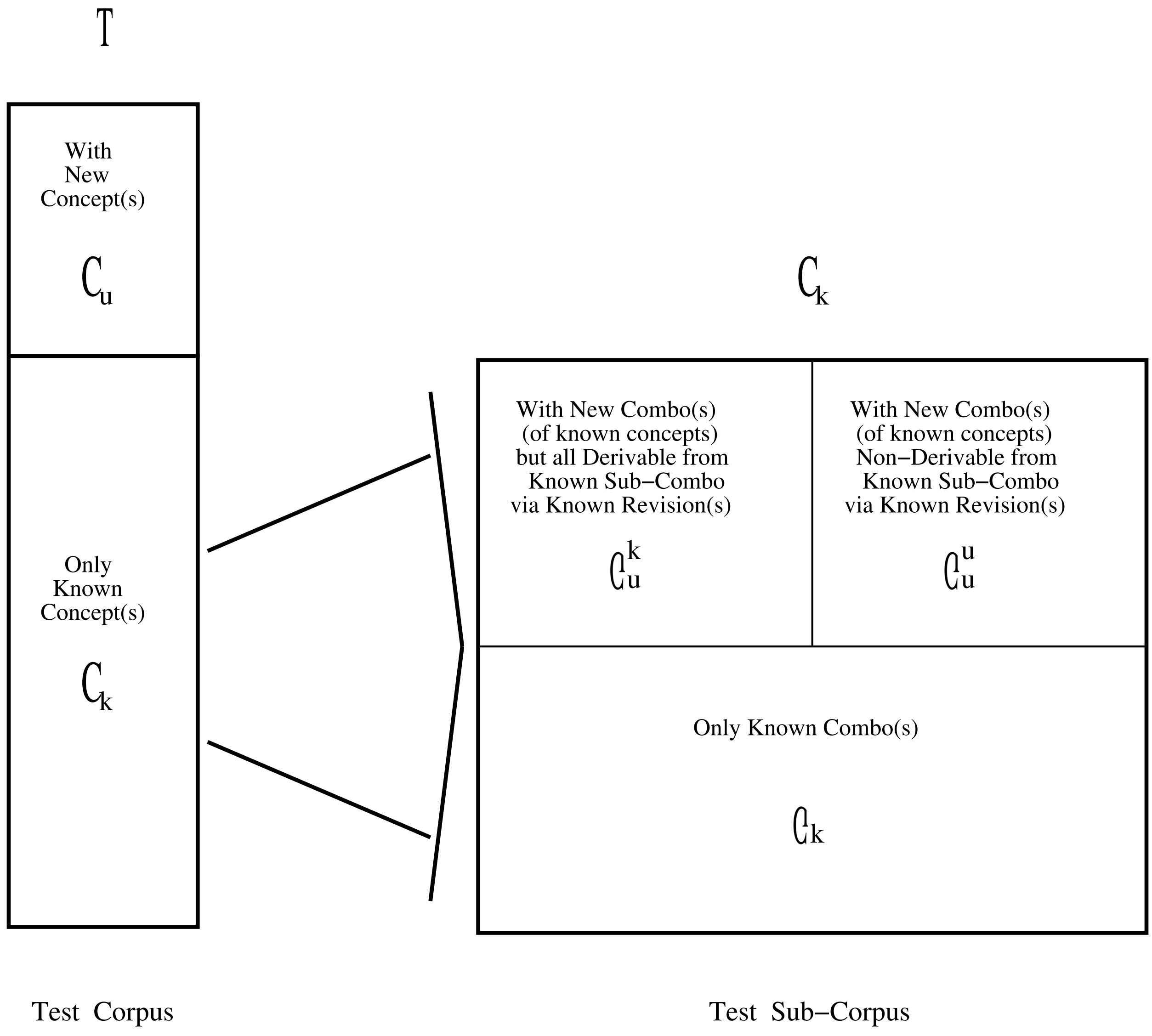

Figure 5.1: Test corpus partitions for conceptual and clustering coverage

- The test corpus contains a sentence S_D where the concept combination $C_D = <$ game-result, winning-streak, losing-streak $>$ is realized by a pattern P_D.
- C_D was not seen in the acquisition corpus.
- But two sub-combinations of C_D, $C_B = <$ game-result, winning-streak $>$ and $C_C = <$ game-result, losing-streak $>$ were seen in the acquisition corpus, respectively realized by patterns P_B and P_C.

In such situations, with one-shot generation, no concept combination rule for C_D would have been abstracted from the acquisition corpus and S_D would thus not be covered. With revision-based generation, the following knowledge structures would have been abstracted from the acquisition corpus:

- A basic realization pattern P_A for the singleton $<$ game-result $>$
- A revision rule R_B to derive P_B from P_A
- A revision rule R_C to derive P_C from P_A

If applying either R_C on P_B or R_B on P_C yields P_D, then S_D can be generated and C_D is thus derivable. Otherwise, S_D cannot be generated and C_D is thus not derivable.

As shown in fig. 5.1, G_u therefore needs to be subpartitioned into:

- the set G_u^k of test corpus sentences clustering concepts in a novel way but using a realization pattern derivable from a known basic pattern via known revision rules,
- the set G_u^u of test corpus sentences clustering concepts in a novel way and furthermore using a realization pattern *not* derivable from a known basic pattern via known revision rules.

Because one-shot generation and revision-based generation use distinct knowledge structures to cluster concepts, a distinct definition of clustering coverage is needed for each. While clustering coverage for one-shot generation was defined (cf. definition 2 above) as the proportion of test corpus sentences conveying *known* combinations of *known* concepts, for revision-based it needs to be defined as the proportion of test corpus sentences conveying concept combinations *derivable* from a known basic combination via known revision rules (among those not conveying any new concepts).

Definition 3 Clustering coverage for revision-based generation

$$V_g^r = \frac{cardG_k + cardG_u^k}{cardC_k}$$

5.2.3.1.3 Paraphrasing coverage Conceptual coverage measures the ability to cope with the appearance of new concepts, while clustering coverage measures the ability to cope with the appearance of new ways to combine known concepts. Paraphrasing coverage has yet another task: measuring the ability to cope with the appearance of novel ways to linguistically express known combinations of known concepts. This is the case for example if the only expression observed in acquisition corpus for the concept combination C_B $= <$ `scoring(player,N,point) , reserve(player)` $>$ was of the realization pattern "PLAYER came off the bench to score N points" and then the alternative realization pattern "PLAYER scored N points off the bench" is seen in the test corpus. Test corpus sentences conveying new concept combinations must therefore be left out of the computation of paraphrasing coverage.

As shown in fig. 5.2, the remaining part of the test corpus, G_k, can then be partitioned in two: P_k comprising the sentences realizing known concept combinations using known linguistic forms and P_u comprising the sentences realizing known concept combinations in a novel way. Paraphrasing coverage for one-shot generation V_p^1, can then be defined as the size of P_k divided by the size of G_k.

Definition 4 Paraphrasing coverage for one-shot generation

$$V_p^1 = \frac{cardP_k}{cardG_k}$$

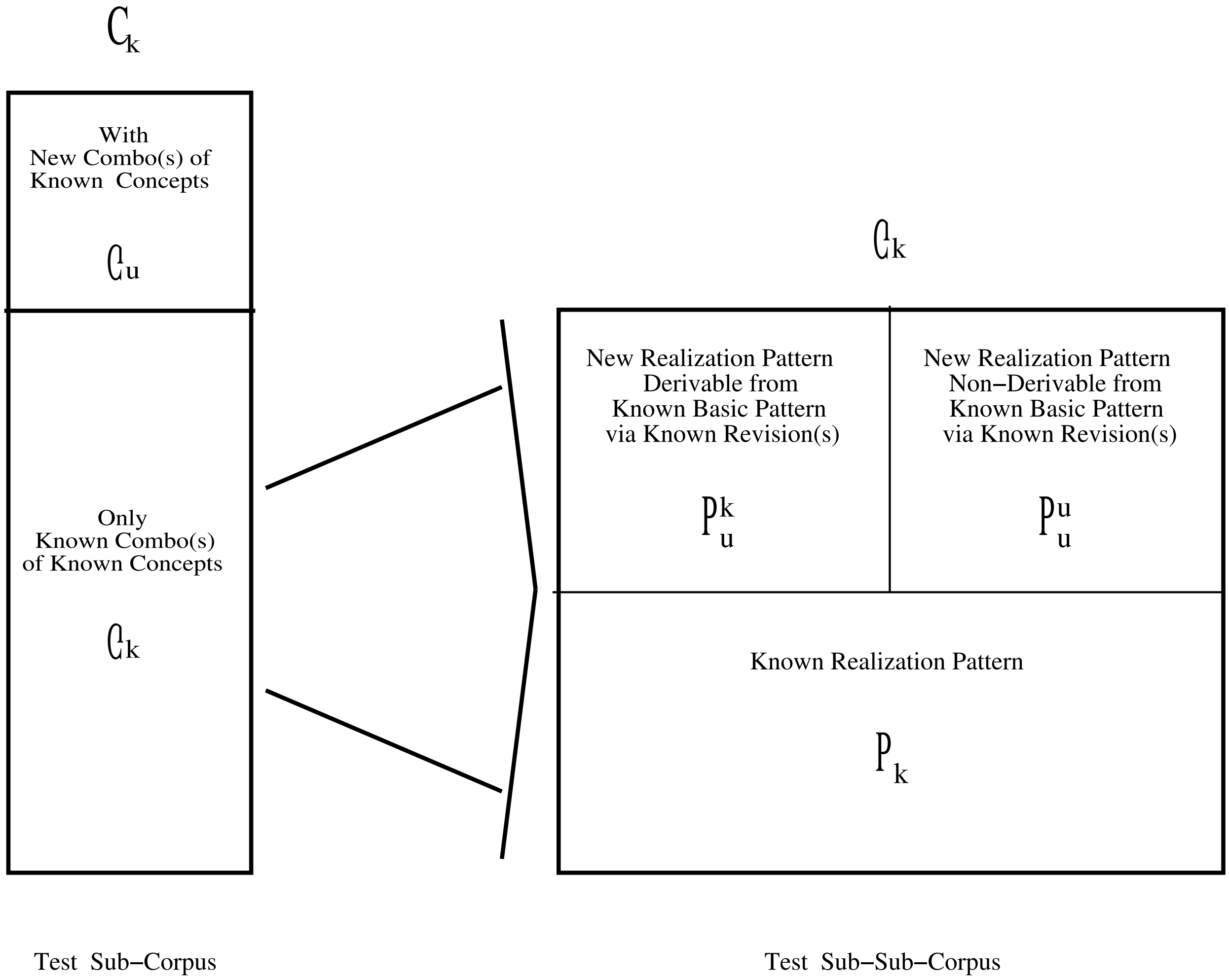

Figure 5.2: Test corpus partitions for paraphrasing coverage

If a realization pattern is known, then, a revision-rule to derive this pattern from a known basic one is also necessarily known. However, an unknown realization patterns may or may not be derivable from a known basic pattern via known revision rules. As shown in fig. 5.2, P_u therefore needs to be subpartitioned into P_u^k comprising sentences with derivable patterns and P_u^u those with underivable patterns. Paraphrasing coverage for revision-based generation V_p^r, can then be defined as the size of $P_k \bigcup P_u^k$ divided by the size of G_k.

Definition 5 Paraphrasing coverage for revision-based generation

$$V_p^r = \frac{cardP_k + cardP_u^k}{cardG_k}$$

5.2.3.1.4 Realization coverage Conceptual, clustering and paraphrasing coverages were deliberately defined independently of each other to evaluate different tasks in the generation process, and identify which task is the bottleneck of the whole process. It is also necessary however, to evaluate the generation process as whole. For this purpose, I define *realization coverage* simply as the proportion of test corpus sentences generable using the knowledge structures abstracted in the acquisition corpus.

In one-shot generation, a sentence is generable iff it conveys a *known* combination of *known* concepts using a *known* realization pattern. Realization coverage for one-shot generation V_r^1 can therefore be simply defined as the size of P_k divided by the size of the test corpus.

Definition 6 Realization coverage for one-shot generation

$$V_r^1 = \frac{cardP_k}{cardT}$$

In revision-based generation, a sentence is generable iff it falls in either one of the following three categories:

- It conveys a *known* combination of *known* concepts using a *known* realization pattern (*i.e.,* it belongs to P_k)
- It conveys a *known* combination of *known* concepts using a realization pattern that is *new* yet *derivable* from a *known* basic realization pattern using *known* revision rules (*i.e.,* it belongs to P_u^k)
- It conveys a combination of *known* concepts that is *new* yet *derivable* from a *known* basic combination via *known* revision rules (*i.e.,* it belongs to G_u^k)

Realization coverage for revision-based generation V_r^r can therefore be defined as the size of $P_k \bigcup P_u^k \bigcup G_u^k$ divided by the size of the test corpus.

Definition 7 Realization coverage for revision-based generation

$$V_r^r = \frac{cardP_k + cardP_u^k + cardG_u^k}{cardT}$$

5.2.3.2 Extensibility coverage

Coverage parameters measure the proportion of the test corpus *sentences* accounted for by the knowledge structures abstracted from the acquisition corpus. In contrast, extensibility parameters measure the proportion of new *knowledge structures* needed to cover the whole test corpus. In other words, coverage estimates how good a job one could do without additional work, whereas extensibility estimates how much more work is needed to do a perfect job.

Having distinguished four types of coverage, I likewise distinguish four type of extensibility:

- *conceptual* extensibility with respect to concepts only.
- *clustering* extensibility with respect to concept combinations only.
- *paraphrasing* extensibility with respect to realization patterns only.
- *realization* extensibility with respect to all knowledge structures.

Collectively, these extensibility parameters differ from the coverage parameters in that they measure proportions of *types* instead of proportions of *tokens*. An uneven distribution of tokens of different types can lead to very different values for the corresponding coverage and extensibility parameters. Suppose for example that 10 concepts were observed in the acquisition corpus and 10 concepts were observed in the test corpus with 5 elements common to both sets. Then, conceptual *extensibility* is only 50%. However if the 5 common elements are also the most common, it is possible that, for example, they are the only concepts in 900 out of the 1000 sentences making up the test corpus. Then conceptual *coverage* is 90%.

Individually each extensibility parameter is defined to mirror the corresponding coverage parameter at the type level. The relation between coverage parameters pictorially represented by fig. 5.1 and 5.2, therefore hold as well for extensibility parameters. In particular:

- Clustering extensibility is defined independently of conceptual extensibility.
- Paraphrasing extensibility is defined independently of both clustering and conceptual extensibility.
- Clustering, paraphrasing and realization extensibility are defined differently for one-shot generation and revision-based generation.

Also likewise coverage parameters, extensibility parameters are defined as the ratio between the cardinal of two sets. Each extensibility parameter differs from its coverage parameter counterpart is that the sets concerned are sets of linguistic structures instead of sets of corpus sentences. The seven extensibility parameters are therefore defined as follows.

Definition 8 Conceptual extensibility

$$X_c = \frac{N_c}{T_c}$$

where:
N_c = *Number of new concepts observed in the test corpus*
T_c = *Total number of concepts observed in the test corpus*

Definition 9 Clustering extensibility for one-shot generation

$$X_g^1 = \frac{N_g}{T_g}$$

where:
N_g = *Number of new combinations of known concepts observed in the test corpus*
T_g = *Total number of combinations of known concepts observed in the test corpus*

Definition 10 Paraphrasing extensibility for one-shot generation

$$X_p^1 = \frac{N_p}{T_p}$$

where:
N_p = *Number of new realization patterns for known concept combinations observed in the test corpus*
T_p = *Total number of realization patterns for known concept combinations observed in the test corpus*

Definition 11 Realization extensibility for one-shot generation

$$X_r^1 = \frac{N_c + N_g + N_p}{T_c + T_g + T_p}$$

Definition 12 Clustering extensibility for revision-based generation

$$X_g^r = \frac{N_r^g}{B + T_r^p}$$

where:
N_r^g = *Number of new revision rules needed to derive from the basic concept combinations, the new combinations of known concepts observed in the test corpus*
and:
B = *Number of basic concept combinations*
T_r^g = *Total number of revision rules needed to derive from the basic concept combinations, all the combinations of known concepts observed in the test corpus*

Definition 13 Paraphrasing extensibility for revision-based generation

$$X_p^r = \frac{N_b^p + N_r^p}{T_b^p + T_r^p}$$

where:
N_b^p = *Number of new basic realization patterns*
N_r^p = *Number of new revision rules*
together needed to derive the new realization patterns for known concept combinations observed in the test corpus
and:
T_b^p = *Total number of basic realization patterns*
T_r^p = *Total number of revision rules*
together needed to derive all the realization patterns for known concept combinations observed in the test corpus

Definition 14 Realization extensibility for revision-based generation

$$X_r^r = \frac{N_c + N_b + N_r}{T_c + T_b + T_r}$$

where:
N_b = *Number of new basic realization patterns*
N_r = *Number of new revision rules*
together needed to derive the new realization patterns observed in the test corpus
and:
T_b = *Total number of basic realization patterns*
T_r = *Total number of revision rules*
together needed to derive all the realization patterns observed in the test corpus

5.2.4 Partially automating the evaluation

Obtaining the evaluation parameters defined in the previous section required repeating - for both test corpora - most of the corpus analysis steps performed on the acquisition corpus, namely:

1. List the domain concept combinations observed in the corpus clusters.
2. List the observed realization patterns of each such combination.
3. Identify the revision tool to produce each such pattern from another simpler pattern.

For the initial acquisition corpus analysis, each step was performed entirely by hand. To avoid repeating this long and tedious process twice over in its entirety, I looked for substeps with potential for automation. During each evaluation round, each step was decomposed into two parts:

- (a) Recognize the test corpus sentences corresponding to usage of a structure (*i.e.*,, respectively a content combination, a realization pattern or a revision tool) already abstracted from the acquisition corpus
- (b) Define and classify new structures for the remaining test corpus sentences.

The first part is a verification task while the second part is a discovery task. The only good candidates for automation are verification tasks: for verification purposes semantic information can be approximated by known lexico-syntactic patterns. In contrast, discovery tasks that involve semantic analysis can hardly be automated: before observing the first lexico-syntactic realization of a semantic message there is no way to encode the various linguistic expressions of that message as a mark of its usage. In our present context, this means that only substep (2a) could be automated.

To address this need, I initiated and supervised the development of CREP [Duford 1993] a system that retrieves in a corpus all the sentences containing a lexico-syntactic pattern specified as a regular expression of words and/or part-of-speech tags. CREP was implemented by Duford. A brief, self-contained presentation of this software tool is given in Appendix D. This appendix also contains detailed, implementation-level examples of its usage to partially automate and speed-up the corpora analyses that underlied both the development of STREAK and the evaluation of the draft and revision generation model on which it is based. In what immediately follows, I explain, at a more intuitive level, the role of CREP for partially automating substep (2a) above for both evaluation rounds.

CREP was first used to compute the proportion of clusters in the first test corpus corresponding to usage of realization patterns abstracted from the acquisition corpus. Each realization pattern was encoded as a CREP expression. Recall from Section 2.4.2, that realization patterns abstract away from domain references, specific lexical items and low-level syntactic variations to capture the mapping from a concept combination onto a particular syntactic structure. They specify the syntactic category used to express each concept and the structural dependencies between these categories. The process of encoding a realization pattern as a CREP expression is presented in Section D.3 of Appendix D.

The resulting CREP expressions were incrementally refined and tested first on the acquisition corpus. It is only after these expressions yielded exactly the same results as those of the manual analysis on the acquisition corpus, that they were run on the first test corpus[9]. For such a systematic run, the CREP package includes a special shell taking as input a file where each expression E corresponding to a given realization pattern, is paired with a file name F. For each $< E, F >$ pair, this shell redirects the test corpus sentences matching E into F. It also redirects the sentences matching none of the expressions into a no-match file. Manual analysis of the no-match file is then required to get the values of the evaluation parameters.

Two systematic CREP searches were independently performed on the first test corpus. One with expressions for statistic cluster realization patterns and the other with expressions for result cluster realization patterns. After these searches, the presence of a cluster in one of the two resulting no-match files has three possible causes:

[9] Some result discrepancies between the manual analysis and the CREP expression runs on the acquisition corpus uncovered errors in the former.

1. The cluster follows a new realization pattern for a concept combination already seen in the acquisition corpus, *i.e.,* it is a member of P_u.
2. The cluster combines concepts individually seen but never clustered together in the acquisition corpus, *i.e.,* it is a member of G_u.
3. The cluster contains a new concept not seen in the acquisition corpus, *i.e.,* it is a member of C_u.

The first step in analyzing each no-match file thus consisted in partitioning it into these three subfiles P_u, G_u and C_u. From the number of elements in each of these files and in the union P_k of all no-match files, the first round values of the coverage parameters for one-shot generation were then derived using definitions 1, 2, 4 and 6[10].

During the process of partitioning each no-match file into the corresponding P_u, G_u and C_u files, the classifications of concepts, concept combinations and realization patterns were extended to the new items encountered in each of these three files. From the number of new elements in each of these three classifications the first round values of the *extendibility* parameters for *one-shot* generation were then derived using definitions 8 to 11.

The extension of these three classifications was also a pre-requisite to the second round of evaluation where the acquisition corpus became the union of original acquisition corpus and of the first round test corpus. For this second round, a new CREP expression - with accompanying extensions in the definition file - was written for each new realization pattern identified during the first round. These expressions were first tested on the new acquisition corpus consisting of the original acquisition corpus plus the first test corpus. Batch CREP runs using both the first year expressions and the new expressions were then performed on the second year corpus. Finally, the no-match files produced by these second round runs were in turn analyzed. This analysis yielded the second round values of both coverage and extensibility parameters for one-shot generation.

Computing the evaluation parameters for revision-based generation, required further analysis of the no-match files, in particular to sub-partition the no-match subfile G_u into G_u^u and G_u^k and the no-match subfile P_u into P_u^u and P_u^k. This partitioning, which needed to be repeated for each evaluation round and each cluster type, involved identifying the surface decrement of each new realization pattern abstracted from the two test corpora. Recall from Section 2.5 that the surface decrement of a realization pattern $\mathcal{P}_{\sqcup}$ is another realization pattern $\mathcal{P}_{\lceil}$ that is syntactically closest to $\mathcal{P}_{\sqcup}$ among all the patterns conveying exactly one less content unit than $\mathcal{P}_{\sqcup}$. The surface decrement of a new pattern identified during either evaluation round was searched for among the final set of realization patterns obtained at the end of the second round. Each new surface decrement pair was then checked against the revision tool abstracted from the acquisition corpus. For this verification, there were four possible outcomes:

1. The new realization pattern resulted from the new application of a revision tool abstracted from the acquisition corpus on a base pattern also abstracted from the acquisition corpus
2. The new realization pattern resulted from the application on *new base* pattern of a revision tool abstracted from the acquisition corpus.
3. The new realization pattern resulted from the application of a *new revision* tool on a base pattern abstracted from the acquisition corpus.
4. The new realization pattern resulted from the application of a *new revision* tool on a *new base* pattern.

The first outcome means that the sentences using this pattern are in G_u^k if they convey a new combination of concepts and in P_u^k otherwise. The last three outcomes mean that the sentence using this pattern are in G_u^u if they convey a new combination of concepts and in P_u^u otherwise. From the number of elements in each of these four no-match subfiles, both rounds of coverage parameters for revision-based generation were then derived using definitions 5 and 7.

During this further analysis of the CREP run no-match files, the classification of both the base patterns and the revision rules were extended to the new items encountered. From the number of new elements in

[10] See Section 5.2.3 for the definition of each evaluation parameter.

	Conceptual	Clustering		
		one-shot	w/ revision	gain
statistic clusters round 1	95.8% (48)	78.3% (46)	97.8% (46)	+19.5%
result clusters round 1	97.6% (85)	96.4% (83)	97.6% (83)	+1.2%
whole sentences round 1	**93.5%**	75.5%	95.5%	+20%
statistic clusters round 2	97.5% (40)	69.2% (39)	97.4% (39)	+28.2%
result clusters round 2	100% (203)	98.5% (203)	100% (203)	+1.5%
whole sentences round 2	**97.5%**	68.2%	97.4%	+29.2%
statistic clusters average	96.6%	73.7%	97.6%	+23.8%
result clusters average	98.8%	97.4%	98.8%	+1.4%
whole sentences average	**95.5%**	**71.8%**	**96.4%**	**+24.6%**
statistic clusters δ	+1.7%	-9.1%	-0.4%	
result clusters δ	+2.4%	+2.1%	+2.4%	
whole sentences δ	**+4%**	**-7.3%**	+1.9%	

Table 5.1: Conceptual and clustering coverage

	Paraphrasing			Realization		
	one-shot	w/ revision	gain	one-shot	w/ revision	gain
statistic clusters round 1	88.9% (36)	97.2% (36)	+8.3%	66.7% (48)	91.7% (48)	+25.0%
result clusters round 1	68.7% (80)	90% (80)	+21.3%	64.7% (85)	85.9% (85)	+21.1%
whole sentences round 1	61.1%	87.5%	+26.4%	**43.2%**	78.8%	+35.6%
statistic clusters round 2	92.6% (27)	96.3% (27)	+3.7%	62.5% (40)	92.5% (40)	+30.0%
result clusters round 2	54.0% (200)	86.0% (200)	+32.0%	53.2% (203)	86.2% (203)	+33.0%
whole sentences round 2	50.0%	82.8%	+32.8%	**33.2%**	79.7%	+46.5%
statistic clusters average	90.7%	96.7%	+6%	64.6%	92.1%	+27.5%
result clusters average	59.3%	88.0%	+28.7%	58.9%	86.0%	+27.1%
whole sentences average	**55.5%**	**85.1%**	**+29.6%**	**38.2%**	**79.2%**	**+41.0%**
statistic clusters δ	+3.7%	-0.9%		-4.2%	+0.8%	
result clusters δ	-14.7%	-11.2%		-11.5%	+0.3%	
whole sentences δ	**-11.1%**	-4.7%		**-10%**	+0.9%	

Table 5.2: Paraphrasing and realization coverage

both these classifications, both rounds of extensibility parameters for revision-based generation were then derived using definitions 12 to 14.

5.2.5 Results

The evaluation results for conceptual and clustering coverage are given in table 5.1 and those for paraphrasing and realization coverage are given in table 5.2. The values for statistic and result clusters for each evaluation round where computed directly from the cardinals of the sets C_u, G_u^k, G_u^u, P_u^k, P_u^u and P_k that partition the test corpus as explained in Section 5.2.3. Next to the value in percent, the absolute number corresponding to 100% is also given. Following the empirically verified independence between clusters (cf. Section 2.4), the value for whole sentences for a given round is defined as the product of the values for each cluster for that round. For each parameter, the average over both rounds and the difference (called δ) between them are also given. For clustering, paraphrasing and realization coverage, the column *gain* indicates the difference between the parameter values for one-shot generation and those for revision-based generation.

The evaluation results for conceptual and clustering extensibility are given in table 5.3 and those for paraphrasing and realization extensibility are given in table 5.4. The values for both statistic and result clusters and each evaluation round were computed directly from the numbers N_c, T_c, N_g, T_g, N_p, T_p, B, T_b^g, N_r^g, T_r^g, N_b^p, T_b^p, N_r^p, T_r^p, N_b, T_b, N_r, and T_r defined in Section 5.2.3.

The values for whole sentences for a given round is defined as the average of the respective percentages for each cluster. Because those percentages were obtained from samples of difference sizes, this average is weighted by the those sample sizes. In other words:

$$W = \frac{S_p S_a + R_p R_a}{S_a + R_a}$$

with:

- W = whole sentence percentage
- S_p = statistic cluster percentage
- R_p = result cluster percentage
- S_a = number of statistic clusters
- R_a = number of result clusters

Unlike for the coverage tables, the gain column values of the clustering, paraphrasing and realization parameters in the extensibility tables were not computed straightforwardly as the difference between the corresponding percentages in the one-shot and with-revision columns. These percentages indicate the proportion of additional knowledge structures needed to fully cover the test corpus with each approach. Since each approach uses different knowledge structures, these proportions have unrelated denominators and subtracting one from the other would not yield a very informative quantity. Instead, the extensibility gain G of revision-based generation over one-shot generation is computed using the following formula:

$$G = \frac{K_1 - K_r}{K_1}$$

with:

- K_1 = number of additional knowledge structures needed to fully cover the test corpus with a one-shot approach
- K_r = number of additional knowledge structures needed to fully cover the test corpus with a revision-based approach

In terms of the numbers used in the other extensibility definitions (cf. Section 5.2.3), K_1 instantiates as:

- N_g for clustering extensibility
- N_p for paraphrasing extensibility
- $N_c + N_g + Np$ for realization extensibility

Similarly, K_r instantiates as:

- $B + N_r^g$ for clustering extensibility
- $N_b^p + N_r^p$ for paraphrasing extensibility
- $N_c + N_b + N_r$ for realization extensibility

In all four result tables, the most interesting values are boldfaced. The first important result is the 38.2% value for the two-round average of the realization coverage parameter for whole sentences and one-shot generation. This result means that the concepts, the half-sentence concept combinations and the half-sentence realization patterns abstracted from the lead sentences over a given year, only account for a little

	Conceptual	Clustering		
		one-shot	w/ revision	gain
statistic clusters round 1	20.0% (10)	41.2% (17)	33.3% (3)	-85.7%
result clusters round 1	9.0% (11)	30.0% (10)	20.0% (5)	-66.7%
whole sentences round 1	14.2% (21)	37.1% (27)	25% (8)	-80%
statistic clusters round 2	10% (10)	61.1% (18)	16.7% (6)	-90.1%
result clusters round 2	0.0% (11)	27.3% (11)	0.0% (5)	-100%
whole sentences round 2	4.8% (21)	48.3% (27)	9.1% (11)	-92.9%
statistic clusters average	15.5%	51.1%	25.0%	-87.9%
result clusters average	4.5%	28.6%	10.0%	-83.3%
whole sentences average	**9.5%**	**42.7%**	17.0%	-86.4%
statistic clusters δ	-10%	+19.9%	-16.6%	
result clusters δ	-9.0%	-2.7%	-20.0%	
whole sentences δ	**-9.4%**	**+11.2%**	-16%	

Table 5.3: Conceptual and clustering extensibility

	Paraphrasing			Realization		
	one-shot	w/ revision	gain	one-shot	w/ revision	gain
statistic clusters round 1	23.1% (13)	25% (4)	-66.7%	30.0% (40)	26.7% (15)	-80.0%
result clusters round 1	65.6% (32)	47.6% (21)	-52.4%	47.2% (53)	37.1% (35)	-50.0%
whole sentences round 1	53.3% (45)	44.0% (25)	-54.2%	39.8% (93)	34.0% (50)	-58.8%
statistic clusters round 2	30% (10)	33.3% (3)	-66.7%	39.5% (38)	33.3% (18)	-85.7%
result clusters round 2	72.3% (83)	40.0% (35)	-76.7%	60.0% (105)	30.4% (46)	-77.8%
whole sentences round 2	67.7% (93)	39.5% (38)	-76.2%	54.5% (143)	31.2% (64)	-79.2%
statistic clusters average	26.5%	29.1%	-66.7%	34.7%	30.0%	-82.85%
result clusters average	68.9%	43.8%	-64.5%	53.6%	33.7%	-63.9%
whole sentences average	**60.5%**	41.7%	-65.2%	**47.15%**	**32.6%**	**-69.0%**
statistic clusters δ	+6.9%	+8.3%		+9.5%	+6.6%	
result clusters δ	+6.7%	-14.3%		+12.8%	-6.7%	
whole sentences δ	**+14.4%**	-4.5%		+14.7%	-2.8%	

Table 5.4: Paraphrasing and realization extensibility

more than a third of the content and linguistic forms of the same sentences the following year. The second important results is the -10% delta value for this same parameter (from 43.2% for the first round down to 33.2% for the second). It means that the sublanguage sample captured by these three knowledge structures over is year or two is nowhere near converging towards the whole domain sublanguage.

What causes this rather low overall coverage? The 95.5% average for conceptual coverage (with a positive delta of +4% from 93.5% after a year up to 97.5% after two) shows that it is not the appearance of new concepts. The domain ontology seems to have been pretty much captured by a single year of analysis. The 71.8% and 55.5% averages for clustering and paraphrasing coverage - both with negative deltas - indicate that one bottleneck is the appearance of new combinations of known concepts and another is the appearance of new linguistic forms to convey known combinations of known concepts. They are both about as significant since while paraphrasing coverage is lower than clustering coverage, the ability of a generator to express a group of concept by various linguistic forms is less crucial than its ability to group concepts in a variety of ways, with at least one linguistic form available for each group.

The revision-based approach, where new combinations and new realization patterns can be derived from simpler ones by applying known revision rules, suppresses both bottlenecks:

- Clustering coverage jumps to 96.4% (a 24.6% improvement from the one-shot approach) and thus becomes even higher than the 85.1% conceptual coverage.
- Paraphrasing coverage jumps to 85.1% (a 29.6% improvement from the one-shot approach).

With a realization coverage at 79.2% (a spectacular 41% improvement from the one-shot approach), its is almost 4/5 of the test corpus sentences whose content and linguistic form are captured by the basic concept combinations, basic realization patterns and revision rules abstracted on the acquisition corpus. The initial intuition than the more compositional revision-based approach would improve robustness, is thus impressively confirmed by the results of the quantitative evaluation of coverage.

The results of the quantitative evaluation of extensibility further confirms the superior robustness of the more compositional approach. With one-shot generation, the average realization extensibility for whole sentences is 47.15%. This parameters measures the proportion of new knowledge structures needed to maintain full-coverage. With revision-based generation it come down to only 32.6%. However, the improvements of the revision-based over the one-shot approach for extensibility, though significant (from a little less than a half to a little less than a third), it is less impressive than for coverage. Even with the revision-based approach almost a third of the knowledge acquisition task must be redone every new year in order to maintain full-coverage of all concept combinations and linguistic forms. This may seem at first a quite a forbidding prospect. Fortunately, such 100% coverage is not needed in most practical applications. Instead, coverage is traded-off against the overhead of further knowledge acquisition. For example, thanks to discrepancies among the occurrence frequencies of the different domain concepts and sublanguage linguistic forms, almost 80% coverage can be attained by implementing only a hard-core of knowledge structures acquired over a single year (with a revision-based approach).

5.3 Quantitative evaluation of portability

Having evaluated the robustness of the corpus analysis results for the domain in which they were acquired in the previous section, I now turn to the evaluation of their portability to a new domain. I first describe the test corpus used for this portability test and the specific linguistic structures that were tested. I then describe the various steps involved in this evaluation and finally discuss the results.

5.3.1 Starting point

The corpus analysis presented in Section 2 resulted in a set of revision tools used in basketball summaries. The goal of this second evaluation effort is to estimate to what degree, these revision tools are also used in another domain. Since these tools were then implemented in the reviser of the generator system STREAK, the results of this evaluation will also indirectly give an idea of STREAK's portability.

As explained in Section 2.2, the basketball domain corpus from which the linguistic structures to evaluate were abstracted, was restricted to report sentences that conveyed no other type of information than:

- Final result and score of the game
- Streak extending (or interrupting) nature of the result
- End of the game statistics (for players or teams)
- Record breaking (or equalling) nature of these statistics

These types were chosen because they were the four most commonly conveyed in the basketball summaries. The portability prospect of linguistic structures like revision tools that have a *semantic* aspect could be estimated only in *quantitative* domains where similar types of information are also common. This is the case of the stock market, meteorology, accounting, labor statistics, etc. As the test domain for this evaluation, I chose the stock market for two reasons: (1) large textual corpora of reports in this domain are available from newswires and (2) it has been the object of several language generation projects [Kukich 1983] [Contant 1986] [Smadja 1991].

The test corpus consisted of reports on the American, Asian and European stock markets by UPI, AP and Reuter compiled from the newsreader. Its size was about 445,000 words. The evaluation involved considering each revision tool abstracted from the basketball corpus, and looking for evidence of its usage in the stock market corpus, in order to compute the proportion of revision tools common to both domains.

As explained in Section 2.2, some revision tools were used for adding both historical and non-historical types of information, while others were used for only one of these two type and not the other. Since incorporation of historical background is another innovative aspect of this thesis, I focused the portability evaluation - like the robustness evaluation - on those tools that are used to add historical information, either specialized or versatile (*i.e.,* I excluded the tools specialized for adding of *non*-historical facts). However, in order to keep the set of revision tools evaluated large enough, I considered as the acquisition corpus, all three years of analyzed basketball summaries: the first year from which the initial set of revisions were identified and the two following years that were used as test corpora for the robustness evaluation and during which the classification of revision tools was extended to account for the new cases encountered.

The final, total classification resulted in a hierarchy of revisions that is given in appendix A. A revision *operation* is the application of a revision *tool* possibly accompanied by a *side* transformation. The upper-levels of the hierarchy classify revision tool applications. The bottom level subdivides applications of the same tool with different side transformations. Depending on whether this bottom distinction is considered or not, the population evaluated for portability consisted of either 37 classes of revision tool applications or 52 classes of revision operations. I estimated the portability at every level in this hierarchy.

5.3.2 Methodology

5.3.2.1 Identifying the source and target realization patterns of each revision tool

Revision tools are abstract transformations operating on linguistic structures. Their usage therefore cannot be directly observed in a corpus. Each tool embodies the structural changes necessary to transform a *source* realization pattern for a given concept combination into a *target* realization pattern for the same concept combination enriched with one additional concept. Usage of a revision tool in a corpus must be indirectly detected by looking for usage of these source and target realization patterns.

The realization patterns whose robustness was evaluated in Section 5.2 were capturing the semantic and syntactic structures of phrase clusters (*i.e.,* roughly of half-sentences, cf. Section 2.4.1 for the exact definition of phrase clusters). In the general case, a revision tool does not apply to the cluster as a whole. Rather it applies locally to one of the cluster's subconstituents. As a result, the same revision tool may apply to many different source patterns and resulting in many different target patterns. Consider, for example, the four corpus clusters below:
C_{S_1}: "Patrick Ewing scored 41 points"
C_{T_1}: "Armon Gilliam scored **a franchise record** 39 points"
C_{S_2}: "Ricky Pierce scored 18 points and Eddie Johnson came off the bench to add 16"
C_{T_2}: "Larry Bird scored 28 points and Brian Shaw came off the bench to add **a season-high** 24"

These four phrases are two examples of what I called surface decrement pairs in Section 2.5.1. More precisely C_{S_1} is a surface decrement of C_{T_1} and C_{S_2} is a surface decrement of C_{T_2}. Even though the realization pattern of the two source clusters C_{S_1} and C_{S_2} differ both semantically and syntactically, the same revision tool, `Adjoin of Classifier` is nonetheless applied to both to obtain the realization patterns of the two corresponding target clusters C_{T_1} and C_{T_2}. This is possible because the revision tool is applied locally to an NP subconstituent whose sub-realization patterns is shared by both C_{S_1} and C_{S_2}.

Because each revision tool has potentially several source patterns and several target patterns associated with it, the first task of this revision tool portability evaluation consisted in compiling for each tool the complete list of its source and target patterns.

5.3.2.2 Abstracting common denominator(s) of source and target patterns

Once realization patterns were properly indexed by revision tools, the next step consisted of comparing, for each tool, all its whole cluster *source* patterns to identify a common denominator. Such common denominators were also identified for its whole cluster *target* patterns, with the additional requirement that it contains the phrase added[11] by the revision[12]. The *signature* of a revision tool can then be defined as the pair: $<$ common source subpattern , common target subpattern $>$. The signature of `nominal-rank adjoin classifier` (with example phrases) is shown in Fig. 5.3.

[11] And displaced in the case of complex revisions.
[12] To make sure that the common target pattern does not reduce to the common source pattern.

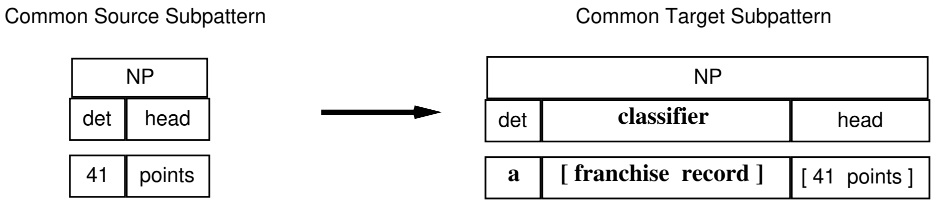

Figure 5.3: Signature of `Adjoin of Classifier` w/ example phrases

1. to lead Chicago to a 95-84 triumph over the Indiana Pacers **that extended the Bulls' win streak to four games**.
2. lifting the Indiana Pacers to a 117 107 victory over Miami **that extended the Heat's losing streak to six games**.
3. powering the Boston Celtics to a 107-99 victory over Cleveland **that snapped the Cavaliers' winning streak at 10 games**.
4. to pace Atlanta to a 116-107 triumph over the Boston Celtics **that snapped the Hawks' slide at three games**.
5. leading the Utah Jazz to a 107 79 victory over Los Angeles **that snapped the Lakers eight game winning streak**.
6. leading New York to a 92-77 victory over the Orlando Magic **that snapped the Knicks ' three game losing streak**.
7. to lead the Cleveland Cavaliers to a 106 103 win over Detroit **that gave the Pistons their third straight defeat**.

Figure 5.4: Example of non-overlapping set of target clusters for a given tool

For some tools, there is no unique subpattern common to *all* its source (or target) whole cluster patterns. In such cases, the signature of the tool has to be defined as a pair of subpattern *lists*, with each element in a list covering one of the maximal disjoint classes of whole cluster realization patterns.

This is the case, for example, of `Adjoin of Relative Clause to Top NP`. One example cluster from the acquisition corpus for each of its target realization pattern is given in Fig. 5.4. No subset of the semantic elements added by the revision (and highlighted in bold) is mapped exactly onto the same syntactic category[13] in *all* seven clusters shown in that figure. For example, while in 1-4 the streak length (underlined) is conveyed by a locative PP, in 5-6 it is conveyed by a classifier and in 7 by an ordinal determiner. Similarly the very fact that the added phrase reports a streak, while conveyed by a noun (*"streak"* or *"slide"*) in 1-6, is conveyed by the adjective (*"straight"*) in 7. The common denominator to *all* the target patterns thus reduces to its source pattern, which cannot constitute alone the signature of the tool. Therefore the target side in this tool signature need to be defined as a list of three sub-patterns, the first covering sentences 1-4, the second sentences 5-6 and the third sentence 7.

In the particular case above, the source side in the signature is a unique pattern. In general, however, both the source and target sides may be lists. Not all pairs that can be formed from these two lists correspond to the application of a revision tool. Therefore, instead of simply two lists of sub-patterns, the general definition of a revision tool signature is a list of < source subpattern , target subpattern > pairs where the source subpattern is a surface decrement[14] of the target subpattern. Usage of the revision tool in the test corpus can then be detected by looking for usage of *both* the source and the target subpattern from any pair in this list.

5.3.2.3 Approximating common sub-patterns as CREP expressions

As *semantico*-syntactic structures the source and target realization patterns of a revision tool are in fact, themselves not directly detectable in an automated way. Following a similar approach as for the robustness evaluation, I approximated realization patterns by *lexico*-syntactic patterns encoded as CREP expressions. This approximation allows partial automation of the search for realization patterns usage in the test corpus. The CREP expressions for the whole cluster realization patterns observed in the two first years of basketball

[13] Recall that a realization pattern captures the mapping between semantic elements and syntactic categories.
[14] cf. Section 2.5.1 for the definition of surface decrement.

Signature = (< source-A1,target-A1 >, < source-A1,target-A2 >, < source-A1,target-A3 >)

```
;; 1st source pattern: team reference followed - at a distance of at least one word -
;; by a determiner, a score, a synonym of ``win'', a synonym of ``over'' and another
;; team reference
SOURCE_A1     TEAM 1- DET 0= SCORE 0= N_WIN 0= OVER 0= TEAM

;; 1st target pattern matching: the 1st source pattern followed - at a distance of at
;; least one word - by ``that'', a verb at the past tense expressing either an
;; extension or interruption, a team reference, a genitive marker, a noun
;; conveying a streak, a preposition and an expression of the streak's length
TARGET_A1     SOURCE_A1 1- that@ 0= (VP_EXTEND|VP_END) 0= TEAM 0= APOST 0= N_STREAK 0= @IN 0= STREAK_LENGTH

;; Variation where the streak length pre (instead of post) modifies the streak noun
TARGET_A2     SOURCE_A1 1- that@ 0= (VP_EXTEND|VP_END) 0= TEAM 0= APOST 0= STREAK_LENGTH 0= N_STREAK

;; Alternative form: a transfer of possession clause
TARGET_A3     SOURCE_A1 1- that@ 0= V_GAVE 0= TEAM 0= POSS 0= ORD 0= STRAIGHT 0= N_STREAK
```

Figure 5.5: CREP expressions approximating the signature of `Adjoin of Relative Clause to Top NP`

reports were available from the robustness evaluation. Using these expressions as a starting point, three tasks remained in order to encode the signature of a each revision tool as a list of CREP expression pairs:

1. Write the missing CREP expressions, *i.e.,* those for whole clusters realization patterns that were observed for the first time in the third year of basketball reports.
2. Now that each tool was associated with a complete list of source CREP expressions and target CREP expressions for the basketball domain, find the common denominator(s) among these expression lists.
3. Group the source and target common denominators into surface decrement pairs. The list of CREP subexpression pairs obtained constituted the *approximate* signature of the revision tool.

Step 2 above - starting from the original CREP expressions encoding whole cluster patterns and getting the CREP sub-expressions approximating a revision tool signature for the same domain, was described in general terms in the previous section. In terms of CREP approximation it translates into two main processes:

- *Sub-expression suppression*, eliminating part of the whole cluster pattern not involved in the revision process.
- *Sub-expression generalization*, factoring out semantic and syntactic details[15] distinguishing between realization patterns but irrelevant at the higher abstraction level of revision tools.

The approximate signature of `Adjoin of Relative Clause to Top NP` is given in Fig. 5.5. The sub-expression definitions appearing in the source and target patterns of this signature are given in Fig. 5.6[16]. These target subpattern match the example sentences of the previous section as follows: TARGET-A1 matches sentences 1-4, TARGET-A2 matches sentences 5-6 and TARGET-A3 matches sentence 7. These expressions factor out low-level semantic distinctions such as streak extension (sentences 1,2,7) vs. streak interruption (sentences 3-6) or winning streak (sentences 1, 3, 5) vs. losing streak (sentences 2, 4, 6, 7). This factoring was done via two sub-expression generalization:

- Grouping WIN-STREAK with LOSE-STREAK, the sub-expressions for noun or noun compounds respectively expressing winning and losing streaks (shown in lines 3, 4, 5 and 8 at the top of in Fig. 5.5).

[15] By adding disjunctions in the sub-expression definition.

[16] Each of these sub-expressions is glossed in comments above its definition to make it understandable without knowing CREP syntax. The reader interested in understanding the detailed CREP encoding of each sub-expression should refer to section D.1 where this syntax is explained.

```
;; Nominals conveying victories
N_WIN          (victory|win|blowout|(blow@ 0= out)|defeat|rout|drubbing|triumph|
               decision|romp|upset)@

;; Nominals conveying streaks
N_STREAK       WIN_STREAK|LOSE_STREAK
WIN_STREAK     win(ning)?@ 0= (streak|spree|flurry|series)@
LOSE_STREAK    (slide@|(losing@ 0= streak@)|((losing@ 0=)? skid@)|drought@|slump@)

STREAK_LENGTH (CARD_PREMOD 1- GAME)

;; Cardinal number in pre-modifying position
CARD_PREMOD    @@@(two|three|four|five|six|seven|eight|nine|ten|eleven|[0-9]+)@@@@

;; Synonyms for game
GAME           (game|decision|outing|contest|session|day)@

;; Ordinal number
ORD            @@@(first|second|third|fourth|fifth|sixth|seventh|eighth|ninth|tenth)@CD|
               (([4-9]|[1-9][0-9])th@)|(1st|2nd|3rd)@@@@

STRAIGHT       (straight|consecutive)@

;; Determiners
DET            ARTICLE|POSS
ARTICLE        a(n)?@|the@
POSS           (my|your|her|his|its|our|their)@ (0= own@)?
APOST          @@@[ ]s@@@@|@@@'s@@@@|@@@'@@@@

OVER           (over|versus|against|of)@

;; Past verbs conveying extensions
VP_EXTEND      (extended|stretched|prolonged|rode)@
VP_END         (snapped|ended|broke|stopped|halted|interrupted)@
V_GAVE         gave@|brought@|handed@|sent@|dealt@
```

Figure 5.6: Definition file for the sub-expressions of Fig. 5.5

- Grouping of VP-END with VP-EXTEND, the sub-expressions for past verbs respectively expressing streak extensions and interruptions (shown in lines 3, 4 at the top and the three lines at the bottom of Fig. 5.5).

The acquisition domain signatures could not be used "as is" on the test domain. Further sub-expression suppressions and generalizations were needed, because the correspondence between the realization patterns in the acquisition and the test domain is an imperfect one. This imperfection is rooted in three different types of discrepancies between the two domains: *conceptual, lexical and rhetorical.* The first occurs when the structure of a concept in the test domain does not exactly match its corresponding concept in the acquisition domain. The second occurs when a concept structure shared by both domains is nonetheless lexicalized differently in each of them. The third occurs when some type of information, though conceptually present in both domains, is absent from the reports of one domain, for no other apparent reasons than some domain-specific rhetorical convention. An example of each types of discrepancies is given in the next subsections describing the precise process of porting a CREP expression.

5.3.2.4 Identifying cross-domain correspondence of conceptual structures

The first task for porting CREP expressions to the test domain is to identify for each conceptual structure of the acquisition domain (basketball) a counterpart structures in the test domain (stock market) At this point it is necessary to briefly review the ontology of the basketball domain presented in Section 2.3. It contained five basic conceptual structures:

- `quality(player|team|game)`, *e.g., "the hot-shooting Boston Celtics"*
- `statistic(player|team,value1,unit)`, *e.g., "Patrick Ewing scored 41 points"*
- `game-result(winner,loser,score)`, *e.g., "the New York Knicks routed the Philadelphia 76ers 106 79"*
- `record(statistic|streak,value,direction,duration)`, *e.g., Armon Gilliam scored* **a franchise record** *39 points"*
- `streak(game-result|record,team,result-type,aspect,length)`, *e.g., "the Boston Celtics won their fourth straight game"*

The stock market domain includes a number of conceptual structures. Only some of them are possible candidates for stock market counterparts of the five basketball conceptual structures. They are:

- `quality(indicator|company)`, *e.g., "the buoyant Hong Kong market"*
- `variation(indicator,direction,value1,value2,unit)`, *e.g., "The Gold Index lost 9.8 points to 2,208.2"*
- `day-result(advances,declines,score)`, *e.g., "advances overpowered declines 814 to 184"*
- `record(variation,value,direction,duration)`, *e.g., "Volume climbed to* **a record** *9.091 billion share"*
- `streak(variation|record,indicator,direction,aspect,length)`, *e.g., "the Dow transportation average retreated for the second straight session"*

Let us examine each basketball conceptual structure in turn to see which stock market structures may correspond to them. `quality` is an example of perfect correspondence between acquisition and test conceptual structures, as illustrated by the pair of examples phrases: *"the hot-shooting Boston Celtics"* and *"the buoyant Hong Kong Market"*. It is also the simplest but least common of all structures.

For `statistic`, the only good counterpart candidate is `variation`. They both share an agentive role, `player|team` for `statistic` and `indicator` for `variation`, as well as a `unit` role. There are two role mismatches: the roles `direction`[17] and `value2` of `variation` have no counterpart in `statistic`. Although

[17] *i.e.,* whether the index went up or down

there were cases of two-valued statistics in the basketball domain, namely shooting performances (*e.g., "Reggie Miller made 13 of 14 shots*"), they were very rare compared to the single-valued ones. In contrast, there were no occurrences of single-valued `variation` in the stock market corpus though they are conceivable[18].

These two mismatches are examples of conceptual discrepancies that do not require adjusting revision tool signatures. This is because most revisions applying to a clause realizing a statistic operate *locally* on the NP realizing the value and unit roles of the statistic. For example, in *Armon Gilliam scored* **a franchise record** *39 points"* the revision is local to the NP *"39 points"*. Thus, when looking for similar revisions in the stock market corpus, the fact that the source clause structure may differ (*e.g., "Armon Gilliam scored 39 points" vs. "The Gold Index lost 9.8 points to 2,208.2"*) is immaterial.

For `game-result`, the immediate candidate counterpart is `day-result`. They both sum-up all the events that took place during the time-slice of the report (a game in the basketball domain, a day or half-day in the stock market domain) and they share an antagonistic role structure, respectively `winner` *vs.* `loser` and `advances` *vs.* `declines`. However, there were no day-result streaks in the test corpus, *i.e.,* no sentence like: "*Advances outnumbered declines* **for the third consecutive days.**"

This absence can be explained as follows[19]:

- The conceptual structures `variation(market)` and `day-result` are totally correlated (if the market goes up then there are more advances than declines and vice-versa); therefore, including streak information for both would be rather redundant.
- The rhetorical convention in the stock market corpus reports is to always convey `variation(market)` up front and `day-result` further down in the report structure; as a result, if a streak is interesting enough to be reported, it is then attached to the former rather than to the latter.

This absence of day-result streaks is an example of *rhetorical* discrepancy between the two domains. Since the present study focuses on game-result *streaks* rather than on *isolated* game-results, the consequence of this rhetorical discrepancy is that a test domain conceptual structure other than `day-result` had to be considered as counterpart to `game-result`, one whose streaks are conveyed in the stock market corpus. The only candidate that satisfies this requirement is `variation`, *e.g.,*
"*the Amex Market Value Index inched up 0.16 to 481.94* **for its sixth straight advance**".

However, there is a role mismatch between `game-result` and `variation`: `variation` is missing the antagonistic `winner` *vs.* `loser` role structure of `game-result`. Instead, whether the agent has won or lost is indicated by a `direction` role: *i.e.,* while a basketball team wins or loses *against* a specific adversary, a financial indicator wins or lose points "on its own". So depending on the value of the `direction` role, the `indicator` role is the counterpart of either the `winner` role or the `loser` role. This is an example of cross-domain discrepancy requiring sub-expression suppression while porting the approximate signatures of the revision tools. Consider again the revision rule `Adjoin of Relative Clause to Top NP` and the 'SOURCE-A1' sub-expression used in its CREP approximated signature (shown on line 2 from the top of Fig. 5.5). The absence of a `loser` role in the financial domain means that porting 'SOURCE-A1' requires deleting its trailing part 'OVER 0= TEAM' which matches the `loser` role in the sports domain.

All the correspondences established so far were for first-order structures. The second-order structures, namely `record` and `streak` take as main argument one of these first-order structures. Their other arguments are the same in both domains. They thus do not introduce further conceptual discrepancies.

5.3.2.5 Acquiring the vocabulary specific to the test domain

Once a correspondence has been established between an acquisition domain concept to a test domain concept, the revision tool signatures involving these concepts need to be adjusted to account for differences in wording.

[18] This absence in the stock market corpus of cases were either the day spread or the final value is reported without the other motivated the view of clauses like *"The Gold Index lost 9.8 points to 2,208.2"* as realizing a two-valued concept as opposed two single-valued concepts sharing the same indicator, direction and unit.

[19] Thanks to Jason Glazier and Karen Kukich for providing insights on stock market domain specifics.

Since in both our domains, most referring expressions use proper names, they constituted the main stumbling block to porting the tool signatures. Names simply do *not* port. The set of CREP expressions was therefore split in three: a first subset for the basketball specific vocabulary, a second subset for the stock market specific vocabulary and a third subset for the shared vocabulary. Definitions for referring expressions were part of the domain specific subsets. While most verbs and common nouns were in the shared subset, elements of some synonymous groups were found in the domain specific subsets. For example, to express the interruption of a streak, the verb *"to rebound from"* is widely used in the stock market domain. In contrast, in the basketball domain *"to break"*, *"to snap"*, *"to halt"*, *"to end"* and *"to interrupt"* are preferred. The vocabulary of the test domain was acquired by running CREP with acquisition domain expressions (written for the robustness evaluation), where the domain references where replaced by wild-cards, "trapping" unknown words between known ones. This technique is discussed both in Section D.1 of Appendix D and in Section 5.3.2 of the CREP manual [Duford 1993].

Porting a CREP expression from one domain to another did not just involve accounting for the conceptual and lexical discrepancies just presented. It also required attention to details like minor rhetorical discrepancies which can prevent CREP expression from matching the desired corpus sentences. Examples of such discrepancies include whether to leave quantity units implicit (*e.g., "New York defeated Philadelphia 106-79" vs. "The Gold Index lost 9.8 <u>points</u> to 2,208.2"*) and punctuation (e.g, the use of "–" *vs.* "," for marking appositions). A full example of porting the signature of a revision tool from the acquisition domain to the target domain using CREP is presented in-depth section D.4 of Appendix D.

5.3.2.6 Manual post-editing of ported CREP expression matches

Because CREP expressions approximate *semantico*-syntactic patterns with *lexico*-syntactic ones, the test corpus sentences that matched the CREP expressions of a revision tool ported signature need to be manually post-edited in order to filter-out the cases where the approximation is not valid.

To illustrate the need for post-editing, consider for example the common target subpattern for the revision tool `Adjunctization of Range into Instrument`. This subpattern reduced to the syntactic mark of the instrument role, namely the preposition "*with*" immediately followed by an indefinite NP[20]. Since CREP is a word and part-of-speech tag regular expression matcher and not a parser, it has no real ability to delimit NPs. The CREP expression for the syntactic mark of the instrument role was thus taken to be simply:

```
with@IN 0= a(n)?@
```

But this expression, does not only match sentences with a genuine instrument PP, such as:
"But futures out-paced the physical market **with a gain in December contracts just after the close of 26 points to 2014** *, extending the premium to a healthy 15.5 points."*

Unfortunately, it also matches sentences where the presence of the bi-gram *"with a"* is purely accidental, such as:
"Traders said the auction results combined <u>with a</u> decline in futures and pressured the cash market lower."
In this second sentence, the NP *"a decline in futures"* is not an instrument adjunct but an object argument of the prepositional verb "*to combine with*". In this case, the *lexico*-syntactic pattern encoded by CREP produces an erroneous approximation of the desired *semantico*-syntactic pattern. The sentences that were wrongly retrieved with this pattern need to be filtered out by manual post-editing. The fact that each match file was analyzed for verification kept this post-editing phase a manageable task, since there was only two such files (one for the chosen source pattern and one for the chosen target pattern) to check for each tool.

[20] Although in the general case, a definite NP can also fill an instrument role, I used the fact that in the basketball corpus all NPs filling the instrument role in the target sentences of `Adjunctization of Range into Instrument` were definite, to simplify the search for corresponding instrument roles in the stock market corpus.

5.3.2.7 Summary of portability evaluation methodology

The methodology I have described in the previous sections to evaluate portability of revision tools to a new domain can be summarized as follows:

1. Define the acquisition domain signature of the revision tool
   (a) Index the whole cluster realization patterns of the acquisition domain by the revision tools for which they are either source or target pattern.
   (b) Decompose the set of whole cluster *source* patterns for the revision tool into maximal subsets sharing a common subpattern.
   (c) Repeat (b) for the set of whole cluster *target* patterns.
   (d) Form a list of surface decrement pairs from the list of common source sub-patterns and the list of common target sub-patterns
2. Considering the surface decrement pairs of the revision tool signature in order, repeat the following steps:
   (a) Find a test domain counterpart to each acquisition domain conceptual structure at play in the *target* subpattern of the pair under consideration.
   (b) Check whether a CREP expression was written during the robustness evaluation for this acquisition target subpattern under consideration. If not write one.
   (c) In this expression, replace the CREP sub-expressions covering acquisition domain proper names by new ones for the corresponding test domain names.
   (d) Run the altered CREP expression on the test corpus. If it matches a test corpus sentence go to step (f). Otherwise proceed to step (e).
   (e) Incrementally suppress, replace and generalize other CREP sub-expressions in the expression - to account for conceptual, lexical or rhetorical discrepancies between the acquisition and test domains - until it matches some test corpus sentence(s).
   (f) Post-edit the file containing these matched sentences. If it contains only erroneous approximations of the sought target subpattern go to back to step (e). Otherwise proceed to step (g).
   (g) Repeat step (a) to (f) with the *source* subpattern of the pair under consideration. If a valid match can also be found for this source pattern, stop: the revision tool is portable. Otherwise resume the entire cycle with the next surface decrement pair in the revision tool signature. If there is no next pair left, stop: the revision tool is considered non-portable.

Substeps 2e and 2f in the above algorithm constitute a generate-and-test approach to approximating realization patterns by CREP expression. Typically, changing or suppressing one CREP sub-expression results in going from too specific an expression with no valid match to either (a) still too specific an expression with no valid match or (b) too general an expression with too many matches to be manually post-edited. The range of expression specificity can only be explored using a trial-and-error process guided by the previous run results. This trial and error process is in fact open-ended: it is always possible to write more complex expressions, manually edit larger match files or even consider larger corpora in the hope of finding a match. So one is left with having to decide, somewhat arbitrarily, that at a given point, the likelihood of finding a match is too small to justify the cost of further attempts. This is why the last line in the algorithm reads "considered non-portable" as opposed to simply "non-portable". The algorithm guarantees the validity of positive results only. In that sense, the figures presented in the next section constitute a lower-bound estimate of the revision tool portability.

5.3.3 Results

In presenting the results of this portability evaluation, I first distinguish between different degrees of portability and give the degree of each revision operation. I then examine how portability is influenced by two factors: the position of the operation in the hierarchy of revisions and the frequency of its usage in the acquisition domain.

5.3.3.1 Degrees of portability

Portability of a revision operation is established when two phrases forming a surface decrement pair (*i.e.,* a pair $<$ source phrase, target phrase $>$) for that operation are found in the test corpus. For example, the two test corpus phrases below:
S_1^t: “*the Hong Kong market*”
T_1^t: “*the* **buoyant** *Honk Kong market*”

which differ only by the describer “buoyant”, form a surface decrement pair establishing the portability of the revision operation `Adjoin of Describer`. This revision operation, allowing the opportunistic addition of qualitative information to a referring NP, was originally identified by the presence of similar surface decrements in the acquisition corpus, such as:
S_1^a: “*the Boston Celtics*”
T_1^a: “*the* **hot-shooting** *Boston Celtics*”

I distinguish three degrees of portability in terms of semantic similarity between:

- The concept C^t that is added from the test source phrase to the test target phrase (in the example above, the current quality of the referred market).
- The corresponding concept(s) C^a that is added from the acquisition source phrase(s) to the acquisition target phrase(s) (in the example above, the quality of the team referred to).

As explained in Section 5.3.2.4, to port each revision operation signature, a correspondence is established from the concepts of the acquisition domain to those of the test domain. Also this portability study was restricted to *historical* acquisition domain concepts such as records or streaks. I thus distinguish between:

- *Same concept portability*: cases where the concept C^t added in the test corpus surface decrement is the exact counterpart of the concept C^a added in the acquisition corpus surface decrement. This is the strongest type of portability.
- *Different historical concept portability*: cases where the concept C^t added in the test corpus surface decrement, while not the exact conceptual counterpart of the concept C^a added in the acquisition corpus surface decrement, is nonetheless a historical concept.
- *Different non-historical concept portability*: cases where the concept C^t added in the test corpus surface decrement is *not* an historical concept. This type of portability spawning the historical/non-historical divide is the weakest.

The two surface decrement pairs above, $< S_1^t, T_1^t >$ and $< S_1^a, S_1^a >$, show that `Adjoin of Describer` is a case of same concept portability. The “current quality of market” fact added by the revision in the test domain is the exact conceptual counterpart of “the current quality of team” fact added by the revision in the acquisition domain.

The two surface decrement pairs below, show that the revision operation `Embedded NP Appositive Conjoin` is a case of portability for different historical concepts:
S_2^t: “*the market is overcoming the weakness in IBM.*”
T_2^t: “*weakness in IBM* **, a market leader and key Dow Component** *, following the announcement of another round of cost-cutting moves will likely weigh on the market.*”

S_2^a: “to lead the New York Knicks to a 97 79 victory over the Charlotte Hornets”’
T_2^a: “to power the Golden State Warriors to a 135 119 triumph over the Los Angeles Clippers **, losers of nine straight games.**”

The transformation from both S_2^t to T_2^t and S_2^a to T_2^a is the attachment of a new fact (boldfaced) as an apposition to an NP embedded in a PP. However, whereas in the *acquisition* domain surface decrement

$< S_2^a, T_2^a >$, this apposition is used to attach a streak information about the embedded NP referent, in the *test* domain surface decrement $< S_2^t, T_2^t >$, it is used to attach another kind of historical fact, namely the past status of the referent.

The two surface decrement pairs below, show that the revision operation `Adjoin Finite Time Clause to Clause` is a case of portability spawning the historical / non-historical divide:
S_3^t: "*Trading volume was a busy 198 million shares.*"
T_3^t: "*Volume amounted to a solid 349 million shares* **as advances out-paced declines 299 to 218.**"

S_3^a: "*to lead the New York Knicks to a 97 79 victory over the Charlotte Hornets.*"
T_3^a: "*to lead Utah to a 119-89 trouncing of Denver* **as the Jazz defeated the Nuggets for the 12th straight time at home.**"

The transformation from both S_3^t to T_3^t and S_3^a to T_3^a is the attachment of a new fact (boldfaced) as a temporal clausal adjunct to the original main clause. However, whereas in the *acquisition* domain surface decrement $< S_3^a, T_3^a >$, this temporal adjunct conveys historical information about the statistic conveyed by the main clause, in the *test* domain surface decrement $< S_2^t, T_2^t >$, conveys another statistic.

Tables 5.5 and 5.6 gives the degree of portability for each of the 53 evaluated revision operations. The first two columns contain the code and name of the revision, the third the number of occurrences in the acquisition domain, and the following four whether is it respectively same concept portable, different historical concept portable, non-historical concept portable or not portable at all.

5.3.3.2 Portability at various levels in the revision operation hierarchy

The revisions whose portability degrees are given in tables 5.5 and 5.6 are the leaves of the revision operation hierarchy presented in Section 2.5[21]. This hierarchy is reproduced here with the following graphical conventions: arcs have four shades of thickness corresponding to the four possible degrees of portability of the revison class they lead to. The thicker the arc, the more portable the class. Only the thinest arcs indicates that the node below them correspond to a non-portable class. Moreover, the nodes corresponding to classes that are portable (at any degree) are boldfaced. These conventions visualize how portability varies with revision specificity as the hierarchy is traversed. They also visualize which classes of revisions are more portable than others.

The hierarchy top-levels are shown in Fig. 5.7. The sub-hierarchies down the nodes `adjoin`, `conjoin`, `absorb`, and `adjunctization` and `nominalization` are respectively shown in Fig. 5.8, 5.9, 5.10, 5.11. The sub-hierarchies down the nodes `recast` and `nominalization` are shown in Fig. 5.12 and the sub-hierarchies down the nodes `demotion` and `promotion` are shown in Fig. 5.13.

Before discussing how portability varies in this hierarchy it is important to remember the different types of distinctions that split the classes at each level into the subclasses at the next level. The hierarchy is seven levels deep. The first level distinguishes between simple revisions that *monotonically* add a new fact to the draft without changing the expression of the facts already conveyed, from the complex revisions that reword the expression of the some existing fact(s) in order to accommodate the new fact into a concise linguistic form. The second level contains the eight main revision tools. Each tool is characterized by a different type of *structural transformation* that the draft undergoes during the revision. Since the simpler revison tools can apply to a variety of syntactic constituents, the third level distinguishes revisions resulting from the application of these tools to different syntactic ranks (*e.g.,* Adjoin to Clause *vs.* Adjoin to Nominal). The next two levels further subdivide revisions in terms of both the semantic role and syntactic realization of the phrase they attaches to the draft[22] (*e.g.,* Adjoin Finite Temporal Clause to Clause *vs.* Adjoin Non-Finite Result Clause to Clause). To add a given content unit, some revision tools can be equally well applied to different draft subconstituents of the same syntactic category but embedded at different levels in the draft

[21] Because the portability evaluation focused on the revision operations that could be used to add historical information, the branches in the hierarchy of Section 2.5 corresponding to revision operations used exclusively in the original corpus to add *non*-historical facts are not shown here.
[22] Or displace in the draft for complex revisions.

Code	Name	Usage	Portable			Non
			sem =	sem ≠		
				histo	¬ histo	
A1	Adjoin Classifier	25	+			
A2	Adjoin Describer	5	+			
A3	Adjoin Relative S to Top NP	1	+			
A4	Adjoin Relative S to Top NP w/ Abridged Ref	20				+
A5	Adjoin Relative S to Top NP w/ Abridged Delete	3				+
A6	Adjoin Relative S to Top NP	9	+			
A7	Adjoin Relative S to Embedded NP w/ Re-Ordering	2				+
A8	Adjoin Non-Finite S to NP	2				+
A9	Adjoin Non-Finite S to NP w/ Abridged Ref	10				+
A10	Adjoin Partitive to NP	1				+
A11	Adjoin Frequency PP to S	13	+			
A12	Adjoin Frequency PP to S w/ Abridged Ref	3	+			
A13	Adjoin Co-Event Non-Finite S to S	12	+			
A14	Adjoin Co-Event Non-Finite S to S w/ Abridged Ref	1				+
A15	Adjoin Co-Event Non-Finite S to S w/ Deleted Ref	4				+
A16	Adjoin Result Non-Finite S to S	4	+			
A17	Adjoin Time Finite S to S w/ Abridged Ref	1			+	
C1	Top NP Appositive Conjoin	5	+			
C2	Top NP Appositive Conjoin w/ Abridged Ref	15				+
C3	Top NP Appositive Conjoin w/ Deleted Ref	4				+
C4	Embedded NP Appositive Conjoin	1		+		
C5	NP Coordinative Conjoin	1	+			
C6	NP Coordinative Conjoin w/ scope mark	2		+		
C7	Top S Coordinative Conjoin	1	+			
C8	Top S Coordinative Conjoin w/ Abridged Ref	5				+
C9	Embedded S Coordinative Conjoin w/ Abridged Ref	1	+			
C10	Embedded S Coordinative Conjoin w/ Abridged Ref	1				+
B1	Absorb NP in S as PP Instrument	1		+		
B2	Absorb NP in S as PP Instrument w/ Abridged Ref	1				+
B3	Absorb NP in S as Co-Event w/ Agent Control	3	+			
B4	Absorb NP in S as Mean w/ Agent Control	1			+	
B5	Absorb NP in S as part of Affected NP Apposition	1				+
B6	Absorb NP in NP as PP qualifier	1	+			
Total		159	14	3	2	14

Table 5.5: Portability degree of simple revision operations

Code	Name	Usage	Portable			Non
			sem =	sem ≠		
				histo	¬ histo	
R1	Recast Classifier as Qualifier	10	+			
R2	Recast Location as Instrument	9				+
R3	Recast Range as Time	1				+
R5	Recast Range as Instrument	1				+
Z1	Adjunctize Created as Instrument	14				+
Z2	Adjunctize Range as Instrument	27	+			
Z3	Adjunctize Range as Instrument w/ Abridged Ref	7				+
Z4	Adjunctize Range as Instrument w/ Deleted Ref	3				+
Z5	Adjunctize Range as Instrument w/ Demoted Agent	1				+
Z6	Adjunctize Location as Instrument w/ Abridged Ref	5				+
Z7	Adjunctize Location as Instrument w/ Abridged Ref	4				+
Z8	Adjunctize Affected as Opposition	1				+
N1	Nominalize w/ Ordinal Adjoin	2	+			
N2	Nominalize w/ Ordinal and Classifier Adjoin	1	+			
N3	Nominalize w/ Ordinal and Qualifier Adjoin	2				+
D1	Demote Affected to Affected Qualifier	2	+			
D2	Demote Affected to Affected Determiner	1	+			
D3	Demote Score to Co-Event Score	1	+			
P1	Coordination Promotion	1	+			
P2	Coordination Promotion w/ Adjunctize	1				+
Total		94	8	0	0	12

Table 5.6: Portability degree of complex revision operations

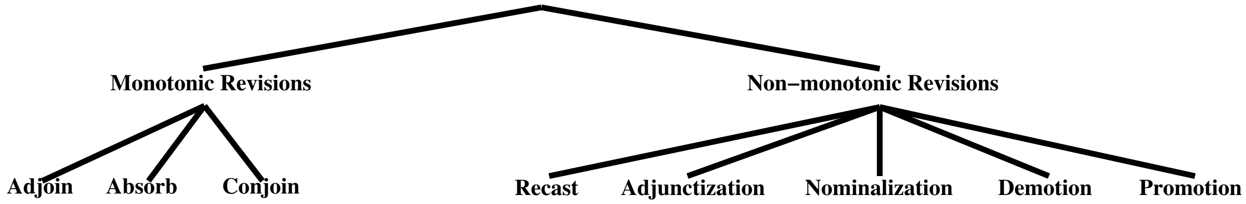

Figure 5.7: Revision operation hierarchy: top-levels

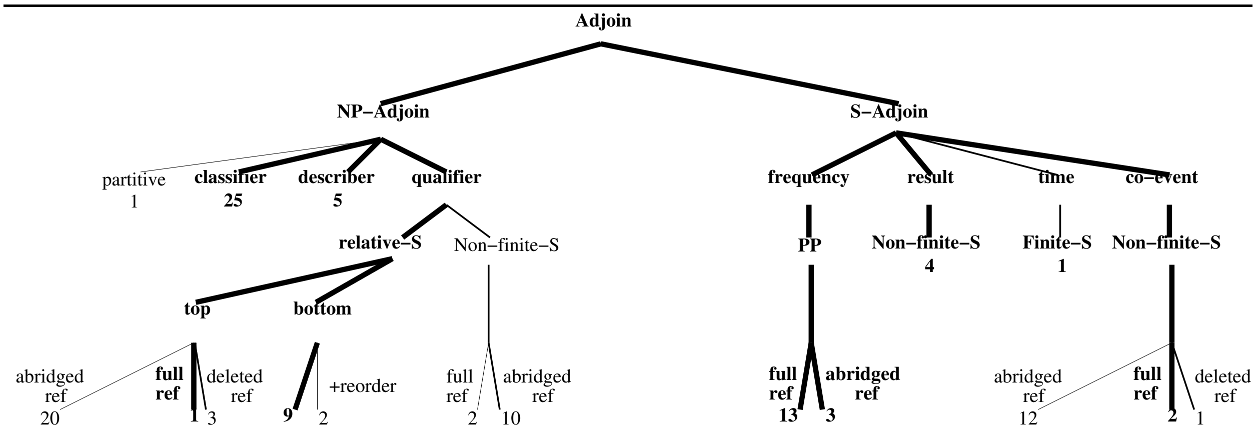

Figure 5.8: Adjoin revision operation hierarchy

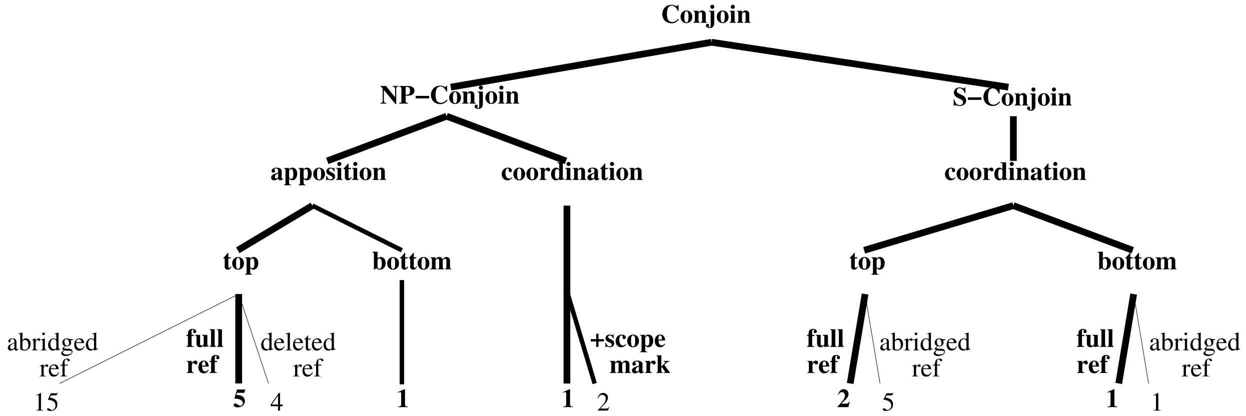

Figure 5.9: Conjoin revision operation hierarchy

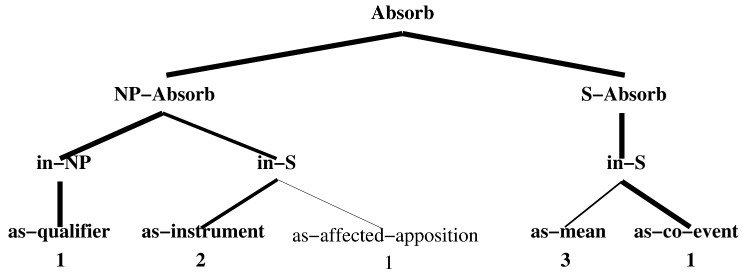

Figure 5.10: Absorb revision operation hierarchy

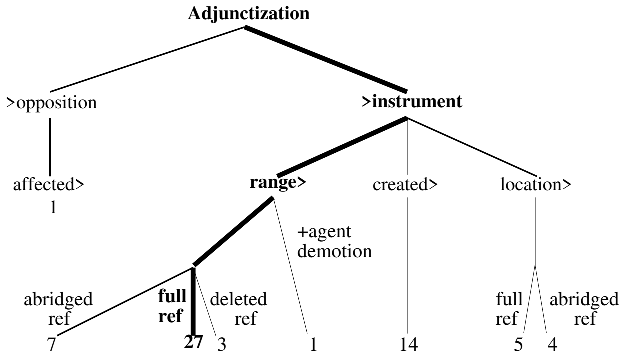

Figure 5.11: Adjunctization revision operation hierarchy

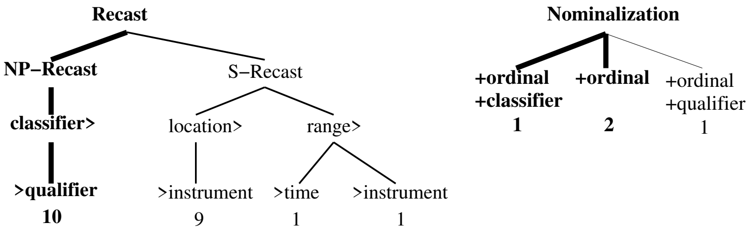

Figure 5.12: Recast and Nominalize revision operation hierarchy

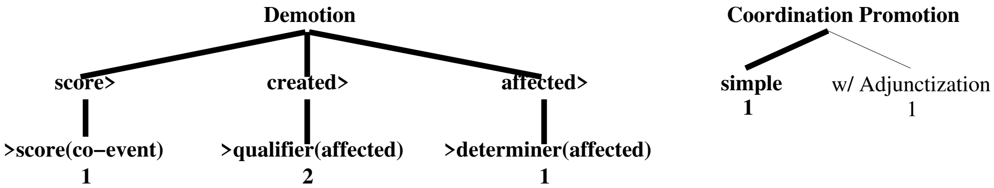

Figure 5.13: Demotion and Promotion revision operation hierarchy

Depth	Nodes	Portable				Non
		sem =	sem ≠		Total	
			Histo	¬ Histo		
Monotonicity	2	100% (2)	-	-	100% (2)	-
Structural transformation	8	100% (8)	-	-	**100%** (8)	-
Draft phrase syntactic rank	12	91.7% (11)	-	-	91.7% (11)	9.3% (1)
Semantic role and/or syntactic realization of added or displaced phrase	31	61.3% (19)	3.2% (1)	6.5% (2)	71.0% (22)	29.0% (9)
Draft phrase embedding depth	35	60.0% (21)	5.7% (2)	5.7% (2)	**71.4%** (25)	38.6% (10)
Accompanying side transformation	53	41.5% (22)	5.7% (3)	3.8% (2)	**50.9%** (27)	49.1% (26)

Table 5.7: Portability from top to bottom of the revision operation hierarchy

structure. For example, to add a streak information on the draft phrase:
"to power the Golden State Warriors to a 135 119 triumph over the Los Angeles Clippers"

the same revision tool `Adjoin Relative Clause to NP` can be applied to the embedded NP referring to the losing team, yielding:
"to power the Golden State Warriors to a 135 119 triumph over the Los Angeles Clippers **, who lost for the ninth straight time.**"

Alternatively, it can be applied to the top-level NP conveying the game result as a whole, yielding:
"to power the Golden State Warriors to a 135 119 triumph over Los Angeles **, that extended the Clippers' losing streak to nine games.**"

The next level in the revision hierarchy thus distinguishes between the applications of these versatile tools at various *embedding depths* in the draft. Finally, at the lowest level, the application of a revision tool is sometimes accompanied by a *side transformation* of the draft, whose object is not add further content but instead to take into account the new presence of the added phrase and avoid repetitions, ambiguity etc. (*e.g.,* the abridged reference *"Los Angeles"* instead of the complete *"the Los Angeles Clippers"* in the last example above, after the addition of a second reference to the team worded *"the Clippers"*.).

In order to quantitatively determine how each of the above distinction affects portability, the percentage of nodes with each portability degree at each level down the hierarchy is given in Table 5.7. The first column indicates the property used to branch the hierarchy at that level and the second number of revision classes at that level. The remaining columns give the percentage (with the absolute number in parenthesis) of those classes which are respectively same concept portable, different historical concept portable, different non-historical concept portable, portable (of any kind) and non portable.

Table 5.7 contains several noteworthy results (in boldface). The first is the striking fact that *all* of the main eight classes of revision tools identified in the sports domain are *same concept portable* to the stock market domain. For everyone of them, I have found at least one phrase pair in the stock market corpus attesting to their usage for adding the exact conceptual counterparts of the sports domain. It is quite impressive that even usage of the revision tools involving the most complex transformations were detectable in the stock market corpus. It is similarly remarkable that even those that were used only a few times in the sports corpus were nonetheless found in the stock market corpus. As the conceptual distance between basketball and finance is fairly large, these result suggests that the generation approach and linguistic data presented in this thesis are likely to apply to other quantitative domains such as auditing, meteorology, cost-analysis or labor statistics.

The second interesting result is that a majority of the very specific revision operations located at the bottom of the hierarchy are portable, even though they distinguish between applications of the revision tools into different semantic, syntactic and rhetorical contexts. Since each such operation is implemented via a revision rule in STREAK, this result suggests that a majority of those rules could be re-used to implement a stock market report generator. As explained in Section 3 the rule base in not a flat list but instead a Functional Grammar (FG) where commonalities between rules are factored out. Each arc in the hierarchy roughly corresponds to a branch in this FG. Since there are 57 portable arcs, and 31 non-portable arcs in the hierarchy, up to two-third of the revision FUG branches developed for a given quantitative domain should be re-usable for the next one.

A third noteworthy result is that among the six properties used for classification in the revision operation hierarchy, only two bring a sharp drop in overall portability. The first of these properties is the semantic role and syntactic realization of the phrase added or displaced by the revision. For example in `Adjoin of` <u>`Clause`</u> `to Clause as` <u>`Time`</u> *vs.* `Adjoin of` <u>`PP`</u> `to clause as` <u>`frequency`</u> the respective distinctive semantic roles and syntactic categories of the added constituent are `time` *vs.* `frequency` and `clause` vs. `PP`. Subdividing the revision set along these lines brings down its overall portability by 20.7%. Let us examine some examples where such distinctions did not port. First consider `Adjoin of Clause to NP`. There are two distinct syntactic forms for such qualifying clauses: relative and non-finite, both used in the sports domain as attested by the following sentences:

1. "*Hakeem Olajuwon scored 18 points and grabbed 17 rebounds Thursday night to pace the Houston Rockets to a 94-83 triumph over the Chicago Bulls* **that completed a two-game sweep**."
2. "*Drazen Petrovic scored 27 points , and Derrick Coleman added 25 points and 12 rebounds Friday night , leading the New Jersey Nets to a 99-85 victory over the Milwaukee Bucks* **, completing a sweep of the season series.**"

In (1), the boldfaced relative clause unambiguously qualifies the game result (lexicalized by the noun "*triumph*"). In the light of the structural and semantic similarity between these two sentences, I also interpreted the boldfaced non-finite clause in (2) as qualifying the game result (this time lexicalized by "*victory*"). Note however, that this second sentence has another potential reading, with the boldfaced non-finite clause modifying the top-level clause conjunction.

In the stock market domain I could find an instance of `Adjoin to NP` for the relative clause case:
3. "*They said the market got an initial boost from overseas markets* **, which proved strong for the second straight day**."

but none for the non-finite clause cases.

There were cases of `Adjoin of Non-Finite Clause`, but they were all modifying other clauses, *not* NPs, *e.g.,*:
4. "*The blue-chip Hang Seng Index , which sank 207.95 points Friday , soared 181.02 points to 9,177.95* **, snapping its three session losing streak**."

Use of such abridged syntactic forms like non-finite clauses whose attachments are not always clear-cut[23] are pervasive in the quantitative summarization sublanguage. In some cases, it may be best to re-consider some interpretation choices based on data from single domain in the light of cross-domain data. Such re-consideration for examples like the one above would have improved the portability results.

The example above was a case where distinctions in terms of the *syntactic form* of the phrase *added* by a revision tool affected its portability. Let us now look at a case where portability is affected by distinction in terms of the *semantic role* of a phrase *displaced* by the revision. This is the case of `Adjunctization Verb Argument into Instrument PP`. In the sports domain, the adjunctized argument could function as a variety of semantic roles including[24]:

- `Created`, where "*Kevin Edwards scored 41 POINTS*"
  becomes "*Kevin Edwards* **tied a career high with** *41 POINTS*"
- `Range`, where "*the Milwaukee Bucks posted A 95 93 VICTORY OVER THE DETROIT PISTONS*"
  becomes "*the Milwaukee Bucks* **snapped a losing streak at five games with** *A 95-93 VICTORY OVER THE DETROIT PISTONS*"

[23] Some corpus sentences are even flatly ungrammatical.

[24] The set of semantic roles used for this analysis are those one used in SURGE, which are themselves derived from various systemic linguistic sources [Fawcett 1987] [Halliday 1985] [Winograd 1983].

- `Location`, where *"the Denver Nuggets rolled to A 124 110 VICTORY OVER THE UTAH JAZZ"* becomes *"the Denver Nuggets* **ended their three game losing streak with** *A 124 110 VICTORY OVER THE UTAH JAZZ"*

In the stock market domain, only cases of adjunctized `Range` roles could be found, *e.g.,*:
"the market posted A 25-POINT REBOUND"
becomes *"the market* **began showing signs of recovery on Monday with** *A 25-POINT REBOUND"*.

Note that all three semantic roles displaced in the sports domain surface as object syntactic roles. Had these cases not been further decomposed in terms of semantic roles, the portability measurement would have improved.

The two examples just presented show that, as soon as semantics is taken into account, portability sometimes depend on gray areas like interpretation choice or analysis granularity. The other cases are mostly due to discrepancies between corresponding conceptual structures in the two domains.

After semantic roles, the other main factor affecting overall portability of a revision tool is the different side transformations that sometimes accompany them. Taking into account these side transformations brings down portability by another 22.6%.

Why is this the case? Consider, for example, the strategy for abridging references presented in section 2.5.2.3. In the example of `Adjunctization of Range into Instrument PP` above, the displaced referring NP is left intact in the revised draft. In the sports domain, there is a variant to this revision operation where the displaced NP is abridged. This variant is illustrated by the example phrases below:
(1) "the Chicago Bulls posted *a 107-100 victory over* <u>*the Minnesota Timberwolves*</u>"
(2) "the Chicago Bulls **remained unbeaten against Minnesota all-time with** *a 107-100 victory over* <u>*the ∅ Timberwolves*</u>."

The revision from (1) to (2) adds a second reference to the losing team worded "Minnesota". To avoid repetition, the initial reference to this team is abridged from "the Minnesota Timberwolves" to only "the Timberwolves". This strategy relies on the fact that in the sports domain most entities have compound names different parts of which can be used unambiguously in different references for the targeted audience, i.e., sports fans. For teams the franchise name and the home city, for players the first and last name etc. This is rarely the case for financial entities. Therefore, this side transformation is much less frequent in that domain, causing the revision operations incorporating them, such as `Adjunctization of Range into Instrument PP with Reference Abridging`, to be non-portable.

Having compared portability at various *depth levels* in the revision hierarchy, I now contrast portability on each *side* of this hierarchy. A quick look at tables 5.5 and 5.6 indicates that the simple revisions on the left side of the hierarchy tend to be more readily portable than the complex ones on the right side. I computed this left/right side contrast at the two bottom levels of the hierarchy. At the very bottom level - *i.e.,* when distinguishing revision operations in terms of both revision tool application and accompanying side transformation - simple revisions were 57% portable compared to only 40% for complex revisions. This difference is not striking. However, at the next level up (*i.e.,* ignoring side transformations) the portability of simple revisions goes up to 85% while those of complex revisions remains no higher than 53%.

Intuitively, this difference seems significant. How likely is it for such a difference to appear at random? A Fisher test[25] comparing the portable/non-portable proportion for simple revisions (17/3) to the corresponding one for complex revisions (8/7) tells us that the probability of this difference being an artifact of our particular set of data is 0.0619. This is very close to the 0.5 probability that is generally accepted as statistically significant. Simple revisions thus indeed seem to be more portable than complex ones. Another Fisher test comparing the portable/non-portable proportion for simple revisions at the next-to-bottom level (19/14) to the corresponding one at the bottom level (17/3) indicates that there is a 0.067 probability that this difference is due to chance. Therefore, when simple revisions do port, it is mostly because the side transformation accompanying it does not. For complex revisions, it is instead mostly because the specific semantic roles affected by the revision do not port.

[25] Thanks to Vasileios Hatzivassiloglou who suggested this test as the most appropriate one for my small sized sample data.

Occurrences	Total	Portable				Non
		sem =	sem ≠		Total	
			histo	¬ histo		
1	13	6	1	2	8	4
2	6	4	1	-	5	1
3	2	2	-	-	2	-
4	1	1	-	-	1	-
5	1	1	-	-	1	-
7	2	1	-	-	1	1
9	2	-	-	-	-	2
10	1	1	-	-	1	-
11	1	1	-	-	1	-
12	1	-	-	-	-	1
14	1	-	-	-	-	1
< 15	31	17	2	2	**21**	**10**
15	1	1	-	-	1	-
16	1	1	-	-	1	-
23	1	1	-	-	1	-
25	1	1	-	-	1	-
38	1	1	-	-	1	-
≥ 15	6	6	-	-	**6**	**0**

Table 5.8: Portability and usage frequency in acquisition corpus

5.3.3.3 Portability and usage frequency in the acquisition corpus

Having examined the influence of the specificity and complexity of a revision operation on its portability, I now examine the influence of another factor: frequency of usage in the acquisition corpus. Just as one would *a priori* expect simple revisions to be, on the average, more portable than complex ones, one would also *a priori* expect the most frequently used ones to be, on the average, more portable than the seldom used ones. Is this indeed the case? Table 5.8 indicates for each occurrence frequency, how many revision operations occurred that many times in the acquisition corpus. It also gives the breakdown of this number in terms of portability degrees. In order to have a significant number of revision operations with more than a few occurrences, the level of decomposition used for this study, was next to bottom of the hierarchy (*i.e.,* distinctions in terms of accompanying side transformations were ignored).

Summing the overall portability for growing ranges of frequencies:

```
[1]     :  61.5%
[2-3]   :  87.5%
[4-7]   :  75.0%
[9-14]  :  40.0%
[15-38]: 100.0%
```

shows that portability clearly does *not* monotonically grow with frequency usage in the acquisition corpus. No clear trend appear before 15 occurrences. Above this frequency, however, all revision operations are portable (even better, they are all *same-concept* portable). Below this threshold, the average overall portability is 67.7%. Intuitively, this difference seems significant. However, a Fisher test comparing the proportion below the threshold (21/10) to the one above (6/0) reveals that the probability of this being an artifact of our small sample (0.162) is too high to consider this difference statistically significant. More data would need to be analyzed to confirm beyond doubt, the intuition that frequently used revision operations are significantly more portable than seldom used ones.

Chapter 6

Related work

There are four main topics of natural language generation that are directly related to the research presented in this thesis:

- Summary report generation
- Generation with revision
- Incremental generation
- Evaluation in generation

Another central theme of this thesis is the generator architecture. However, because a great number of generators have been implemented and their variety in terms of architectures is only matched by their variety in terms of applications, I cannot comprehensively review generator architectures in the present thesis. For such a review with a special focus on the relation between architecture and lexical choice, see [Robin 1990]. In this section, I mention architecture issues only for systems either whose application (summary report generation) or approach to generation (revision-based or incremental) is directly related to those of STREAK.

6.1 Related work in summary report generation

Prior to STREAK, six main systems generated natural language reports to summarize quantitative data: ANA [Kukich 1983] [Kukich 1985], FOG [Bourbeau *et al.* 1990] [Polguere 1990], GOSSIP [Carcagno and Iordanskaja 1993], LFS [Iordanskaja *et al.* 1994], SEMTEX [Roesner 1987], and SAGE [Roth *et al.* 1991]. I briefly review each of these systems in the following subsections.

Other generation systems that produced reports are not directly relevant to the present thesis, because the reports are not specifically summaries and because they worked in qualitative domains (*e.g.,* Danlos' generator [Danlos 1986], PAULINE [Hovy 1988], Kalita's system [Kalita 1989], TEXPLAN [Maybury 1990]). While there are systems that attempt to summarize textual input such as newswire articles by selecting representative sentences in the text (*e.g.,* [Rau 1987]), they are not directly relevant either because they involve no generation.

6.1.1 Ana

The field of summary report generation was pioneered by Kukich. Her system ANA [Kukich 1983] summarizes the daily fluctuations of several stock market indexes from half-hourly updates of their values. A report generated by ANA is given in Fig. 6.1. Internally ANA consists of a pipeline of four components (the first written is C and the remaining three in OPS5):

Thursday June 24, 1982
wall street's securities markets meandered upward through most of the morning, before being pushed downhill late in the day yesterday. the stock market closed out the day with a small loss and turned in a mixed showing in moderate trading.
the Dow Jones average of 30 industrials declined slightly, finishing the day at 810.41, off 2.76 points. the transportation and utility indicators edged higher.
volume on the big board was 558,600,000 shares compared with 627,100,000 shares on Wednesday. advances were ahead by about 8 to 7 at the final bell.

Figure 6.1: An example report generated by Ana (from [Kukich 1983])

- *The fact generator* which compiles the half-hour updates of the stock market indexes into facts. Facts are aggregates of various statistics over a given period of time. They are represented by OPS5 Working Memory Elements (WMEs). An example fact is shown in Fig. 6.2.
- *The message generator* which groups facts into clause-sized content units called messages and also represented as OPS5 WMEs. An example message is shown in Fig. 6.2.
- *The discourse organizer* which groups and orders the messages into paragraphs. It also collapses together messages which share several features (*e.g.,* when two consecutive messages differ exclusively with respect to their subjects, these messages get collapsed into a single message with a compound subject).
- *The text generator* which goes over the ordered message list produced by the discourse organizer, maps each message onto a clause, and combines clauses together into complex sentences.

In ANA, the text generation subtasks defined in section 3.1 are thus distributed among the above components as follows:

- Content production and selection are performed by the fact and message generators.
- Discourse level content organization is performed by the discourse organizer.
- Phrase level content organization, lexicalization, semantic grammaticalization, morpho-syntactic grammaticalization and linearization are all performed by the text generator.

The tasks performed by ANA's text generator thus correspond directly to the tasks implemented in STREAK[1]. ANA's text generator relies on two key knowledge sources: (1) a phrasal lexicon that defines direct mapping from messages to hand-coded predicate clause patterns and subject nominal patterns and (2) a set of rules defining how phrasal entries can be combined to form multiple clause sentences that fluently combine several facts. An example of phrasal entries is given in Fig. 6.3. Such entries comprise up to eight words. Each output sentence is built by assembling two or three of such phrasal entries.

6.1.2 Systems based on the Meaning-Text Theory

Three important summary report generation systems have been built in recent years by the same research team distributed in three cites: Odysee Research Associates, University of Montreal and Cogentex. The systems all use the Meaning-Text Theory (MTT) [Mel'cuk and Pertsov 1987] as underlying linguistic model.

The first of these system is FOG [Bourbeau *et al.* 1990]. It produces daily local marine weather bulletins, in both English and French, from meteorological measurements. An example English report generated by FOG is given in Fig. 6.4. As opposed to most other[2] generation systems which are research prototypes, FOG

[1] Except that STREAK has the option not to express all the messages it is given in input when all of them cannot fit in a readable sentence. It is thus also performing final content selection. In constrast, the text generator of ANA has no such option. It must convey all the element in its input message list.

[2] In fact, probably even all others except PLANDOC [Kukich *et al.* 1994].

```
Example of fact: (make fact ^fact-name half-hour-stat
                            ^indicator-name Dow-Jones-industrial-average
                            ^indicator-type composite
                            ^date 04/21
                            ^time 11:30am
                            ^current-level 1192.82
                            ^direction down
                            ^degree 2.03
                            ^cumulative-direction up
                            ^cumulative-degree 1.35
                            ^high-or-low-mark nil)

Example of message: (make message ^date 6/24
                                  ^topic Dow-Jones-industrial-average
                                  ^subtopic Dow-Jones-industrial-average-status
                                  ^subject-class Dow-Jones-industrial-average
                                  ^direction down
                                  ^degree small
                                  ^time closing
                                  ^variable-level 810.41
                                  ^variable-degree 2.76)

Corresponding sentence: ``the Dow Jones average of 30 industrials declined slightly,
finishing the day at 810.41, off 2.76 points''
```

Figure 6.2: Example fact and message in ANA (from [Kukich 1983])

```
(make phraselex ^phrase-type predicate
                ^topic general-market
                ^subtopic interesting-market-fluctuation
                ^duration long
                ^degree small
                ^direction up
                ^time early
                ^subject-type name
                ^subject-class market
                ^verb-past-singular-inflection "meandered"
                ^verb-past-plural-inflection "meandered"
                ^verb-present-participial-inflection "meandering"
                ^verb-infinitive-inflection "to meander"
                ^predicate-remainder "upward through most of the morning"
                ^random 5
                ^length 14)
```

Figure 6.3: Example phrasal lexicon entry in ANA (from [Kukich 1983])

Winds nothernly 20 to 30 knots becoming northwest Wednesday morning. Snow and rain tapering to flurries Wednesday morning. Visibility zero to 3 in precipitation. Temperatures near zero.

Figure 6.4: An example report generated by FOG (from [Polguere 1990])

The system was operating for 6 hours 36 minutes and 57 seconds. Usage was particularly intense between 16:32:03 and 18:54:29 with idle time only 27 cycles during this period. Seven users worked on the system. Five of them used mostly compilers (C,Lisp,Fortran) and the Prolog interpreter. VLADIMIR and LEO read numerous files. VLADIMIR was interested in system priority tables. LEO listed many user files from his own group. He initiated large print jobs using these files. VLADIMIR failed to change access parameters for system files. No modification to system files were noted.

Figure 6.5: An example report generated by GOSSIP (from [Carcagno and Iordanskaja 1993])

is a finished product in everyday use at weather centres in Eastern Canada. Compared to ANA, in terms of conceptual summarization, FOG averages input data along spatial dimensions in addition to temporal ones. In terms of linguistic summarization, it relies on the telegraphic style peculiar to weather forecasts instead of relying on the clause combining journalistic style typical of newswires. The second of these systems is GOSSIP [Carcagno and Iordanskaja 1993] which summarizes the activity of computer users from the audit trail produced by the operating system of the machine they are logged on. Its input is intended to assist the system administrator in detecting system usage that is suspicious from a security standpoint. An example of such a summary is given in Fig. 6.5. As opposed to FOG, GOSSIP is only a monolingual research prototype. The latest system in this line of work is LFS, which generates bilingual reports[3] summarizing Canadian labor market statistics.

These three systems are all implemented using an object-oriented extension of PROLOG and share the same architecture. This architecture consists of a pipeline of three components:

- A text planning component.
- A lexicalization component.
- A linguistic realization component.

Multi-lingual generators based on this architecture have a single text planning component but a distinct pair of lexicalization and linguistic realization components for each output language.

The text planner is based on the notion of *topic tree*. A topic tree represents a stereotypical textual organization pattern followed by the reports in a given application domain. It is comparable to one particular traversal of a textual schema [McKeown 1985]. The nodes of a topic tree are domain concepts and its arcs are very general relations which can hold between concepts in many domains (*e.g.,* `object, aspect, subaction, attribute, element`). Each node in the topic tree is represented as an PROLOG "object". The fields of the object encode both the concept represented by the node and the arcs linking the node to the other nodes in the topic tree. The methods of the object encapsulate the various text planning procedures that can concern the concept represented by the node. Text planning starts by a first top-down traversal of the topic tree. When a node N with concept C is visited, the methods of N are executed with the following possible effects:

- Instantiation of C into a fact from the input statistic database (if there is an instance of C in the input database).
- Deletion of the N from the topic tree (if there is *no* instance of C in the input database).

[3] Also in English and French.

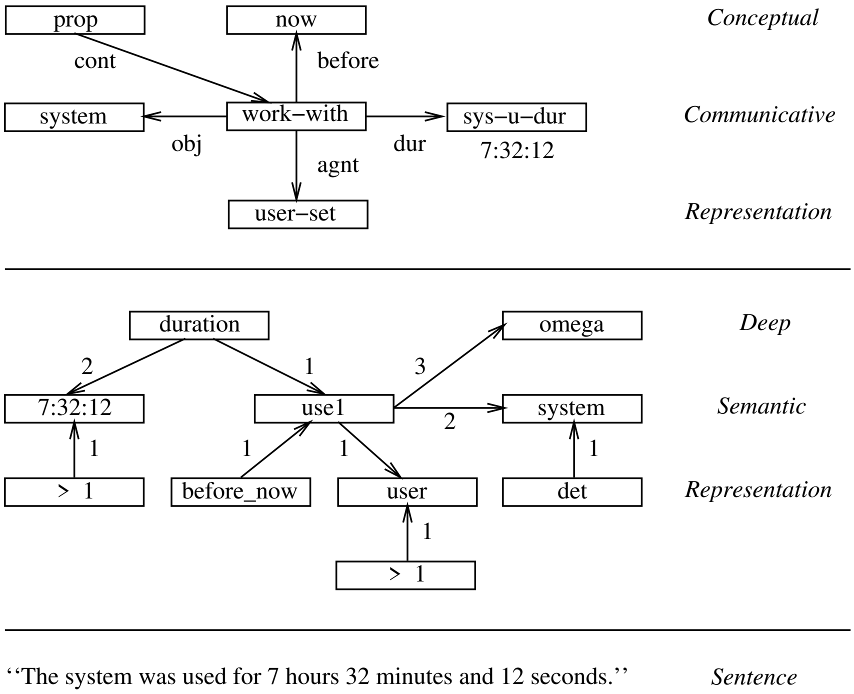

Figure 6.6: Conceptual Communicative *vs.* Deep Semantic representations the Meaning-Text Theory (from [Carcagno and Iordanskaja 1993])

- Addition to the topic tree of new arcs and nodes added to the topic tree below N (these additions implements domain reasoning allowing the text planner to deduce additional facts from those present in the input).

After this first traversal is completed, the resulting *instantiated* topic tree is again traversed top-down. Two main tasks are performed during this second traversal: arc merging and node ordering. If several arcs down a given node share the same label and lead to nodes containing instances of the same concept, these arcs are merged into a single arc leading to a single node containing the list of instances. Linguistically, such merging operations ultimately translate into replacement of a verbose conjunction at the clause rank (*e.g., "[Vladimir read numerous files] and [Leo read numerous files]"*) by a concise conjunction at the nominal rank (*e.g., "[[Vladimir] and [Leo]] read numerous files"*). After arc merging, each node contains a sentence sized content chunk. Each node is then assigned a number corresponding to the sentential position in the output text. At the end of this second traversal, each node in the topic tree contains a flat conceptual network, called a *Conceptual Communicative Representation* (CCR) encoding the content of one report sentence.

The CCR of each topic tree node is then passed on to the lexicalization components which accesses a dictionary and produces a corresponding *Deep Semantic Representation* (DSemR). The DSemR is the input to the linguistic component. Compared to the CCR, the DSemR is also a flat network but in which nodes are no longer language-independent concepts but instead word senses of a specific language. An example CCR together with the corresponding English DSemR is given in Fig. 6.6.

The task of the linguistic component is to generate a natural language sentence expressing the content encoded in the DSemR. The linguistic component is an implementation of the Meaning-Text Theory (MTT) [Mel'cuk and Pertsov 1987]. The MTT is a lexicalist, stratificational, declarative and generation-oriented linguistic theory where sentences are represented at five distinct layers:

- *Deep SEMantic Representation* (DSemR)[4] is a flat network whose node are word senses which are atomic within the theory, *i.e.,* which cannot be further decomposed in terms of other word senses. The DSemR of a sentence represents the *whole* recoverable meaning of the utterance, whether explicitly conveyed or not. The DSemR may also include an annotation specifying the *theme vs. rheme* partition of the sentence.
- *Surface SEMantic Representation* (SSemR)[5] is also a flat network whose node are word senses. It differs from the DSemR in that (1) it may comprise non-atomic word senses, (2) it represents only the *explicit* content of the sentence and (3) it indicates which node corresponds to the main verb. The mapping from DSemR to SSemR thus involves reducing some subnets into single nodes labelled by semantically rich words, dropping nodes corresponding to content best left implicit and chosing which explicit content unit should head the linguistic structure.
- *Deep SYNTactic Representation* (DSyntR) is a tree whose structure represents the constituency and dependency relations inside the sentence. The nodes of the tree are either open-class words or *lexical functions.* The MTT comprises about 50 such functions to model common semantic relations between two words. There are syntagmatic lexical functions such as `Oper1` relating a noun to its agent oriented support verb (*e.g.,* `Oper1`("rebound") = "to grab"[6]) as well as paradigmatic lexical functions such as **Anti** relating antonyms (*e.g.,* `Anti`("victory") = "defeat"). The arcs of a DSyntR are either numbers distinguishing the various elements in a word argument structure, `attr` for any type of attributive relation, `coord` for any type of coordinated elements and `append` for any type of apposited, parenthetical or comment elements.
- *Surface SYNTactic Representation* (SSyntR) is a tree that matches the actual syntactic structure of the sentence. The SSyntR include nodes for the closed-class words in addition to the open-class word nodes that it inherits from the DSyntR. In the SSyntR, the lexical function nodes have been replaced by subtrees corresponding to their values. Finally, the arcs of an SSyntR are *syntactic* roles such as *Subject*, *Object*, *Determiner* etc.
- *MORPHological Representation* (MorphR) is an ordered list of words, each annotated with all the morphological features necessary to compute its inflection.

The Conceptual Communicative Representation (CCR) of MTT-based systems corresponds to the Deep Semantic Specification (DSS) of STREAK[7]. Therefore STREAK covers the generation subtasks carried out by the lexicalization and linguistic realization components in MTT-based generators. However, there is no direct correspondence between the sentence representation layers of the MTT and those of STREAK.

Mapping a flat conceptual network onto a natural language sentence involves four main subtasks:

- Choosing which element in the net to explicitly convey in the sentence (explicitation)
- Choosing how to group elements into constituents and choosing the structural dependencies among these constituents (hierarchization)
- Mapping the some of the chosen elements onto open-class words (lexicalization)
- Mapping the remaining chosen elements onto syntactic features and closed-class words (grammaticalization)

In STREAK, explicitation and hierarchization are performed first, followed by lexicalization and then grammaticalization. In the MTT, it is lexicalization[8] that is performed first, followed by explicitation and hierarchization and finally grammaticalization. What is gained by lexicalizing the whole conceptual input before deciding which part of it to convey explicitly remains unclear. Note that the system FOG differs from

[4] Sometimes simply called SEMantic Representation (SemR).

[5] Sometimes called Reduced Semantic Represenation (RSemR).

[6] In general, lexical functions have multiple values, so for example an other possible value for `Oper1`("rebound") is "to haul in".

[7] cf. section 3.12 for the definition of STREAK's internal representation layers.

[8] At least most of it. Lexicalizations resulting from application of lexical functions is performed after explicitation and hierarchization.

both GOSSIP and LFS in that it does not make use of the two semantic representation layers and directly maps CCR networks onto DSyntR trees. One advantage of early lexicalization is that word argument structure can guide the hierarchization process. But aside from verbs, only a few words take arguments. Moreover, adverbial relations linking most clauses inside complex sentences are not lexically constrained. Thus, at the complex sentence rank, content organization options are largely underconstrained by lexical choices. It thus seems that the lexicalist flavor of the MTT is more motivated by its origin in lexicographical work than by its particular suitability to sentence generation.

6.1.3 Other summary report generators

Another system implemented for the domain of labor market statistics, this time in German, is SEMTEX [Roesner 1987]. It produces summaries of similar content and style to those generated by LFS. It is also implemented in an object-oriented fashion (it uses FLAVORS, a LISP-based object system whereas LFS uses a PROLOG-based object system.). However, in terms of architecture, SEMTEX is far closer to ANA than to LFS. It consists of a pipeline of three components:

- AMTEX, which queries an input labor statistic database and produces case frames, each one representing a fact to convey in the report. Which types of facts get produced by AMTEX depend on parameters defining how statistical values should divided into various population subgroups. Case frames are encoded as "objects" and comprise the following fields: `direction` (with values increase, decrease or unchanged), `quantity, time-period, manner, from, by` and `to`.
- SEMSYN, which relies on a phrasal lexicon to map each element in the list of case frames it receives from AMTEX onto a corresponding partially lexicalized syntactic tree.
- SUTRA-S, which produces a German sentence from the partially lexicalized syntactic tree it receives from SEMSYN.

SEMTEX relies on coordination with ellipsis as its main linguistic summarization device.

The SAGE system [Roth *et al.* 1991] generates a combination of text and charts summarizing the current status of a large engineering project that help managers quickly respond to unforeseen difficulties or missed deadlines and keep the project on track. This system suggests the great potential for office-automation applications of summary report generation. However, while focusing on the issue of coordinating textual and graphical media, SAGE relies on ad-hoc techniques for text generation (cf. [Roth *et al.* 1991], p.216).

6.1.4 Comparison with STREAK

An important first difference between previous summary report generation systems and STREAK is that they perform two tasks that are not implemented in the current version of the STREAK prototype: conceptual summarization and combination of multiple sentences in a paragraph. These tasks are the responsibility of the fact generator and discourse planner in the general generation architecture presented in Fig. 3.13 of section 3.3. STREAK focuses on content realization and phrase level content planning bringing about several significant improvements over previous systems with respect to these two tasks. In particular, STREAK:

- Reports facts in their historical context.
- Trades off informativeness and conciseness against readability.
- Generates more complex sentences.
- Scales up better.
- Encodes more paraphrasing power.

I briefly elaborate in each of these points in the following sections.

Historical context The most striking improvement of STREAK compared with previous summary generation systems is that it systematically provides the historical background of the new events it relates. The reports generated by STREAK thus not only *summarize* a basketball game but also *contextualize* it. With the exception of ANA, previous systems summarized their input statistics out of context. ANA does handle a very restricted class of historical information[9]: all-time highs and lows. ANA detects such extrema by having the all-time high and low values of each index encoded in its content determination rules. Since every time a record is broken, these values need to be updated by hand in the rules in which they are used, this treatment of historical information is *ad-hoc*. It could not be extended to the variety of records and streaks conveyed in the summaries generated by STREAK.

Trading off conflicting goals The great originality of STREAK's architecture is that it allows the final decision of whether to include a complementary fact in the report to be made *opportunistically*, under surface form constraints. This is in contrast with any previous summary generation system. It makes possible explicit trade-offs between the three inherently conflicting goals of summarization: (1) maximize information, (2) minimize space and (3) maximize readability. This type of trade-off is illustrated in the first example run of section 4.4 when STREAK rejects several candidate facts for inclusion in the lead sentence of the report on the basis of corpus observed limits on sentence complexity. A system where all the facts to convey are decided before any of them is linguistically realized cannot make that kind of decision. This was the case for all previous summary generators.

Sentence complexity STREAK generates more complex sentences than any other generation system. They are more complex in that they convey more propositions, are syntactically deeper and contain more words. The table below, compares the maximum factual density, syntactic depth and lexical length in sentences from example reports generated by three previous summary report generators and STREAK. It shows a clear difference in the overall complexity between the sentence generated by previous systems and those generated by STREAK.

	GOSSIP	FOG	ANA	STREAK
Factual density	5	3	4	12
Syntactic depth	5	4	6	10
Lexical length	17	25	34	45

These figures are estimates that I made by looking at the few example reports provided in publications about these systems[10]. For lexical length, I counted the total number of words, both closed-class and open-class. For syntactic depth, I parsed the deepest sentences by hand. For factual density, I represented the content of the most informative sentences as a set of predicate calculus formulas. For example, the content of the third sentence in Fig. 6.5 can be represented by the five predicates below and thus conveys 5 facts[11]:
use(user-1,c-compiler,mostly,period-1)
use(user-2,lisp-compiler,mostly,period-1)
use(user-3,fortran-compiler,mostly,period-1)
use(user-4,prolog-interpreter,mostly,period-1)
use(user-5,lisp-compiler,mostly,period-1)

As explained in section 1.1.2.4, it is the draft and revision approach of STREAK that allows it to pack in so many facts in such complex sentence structures. Planning and realizing the same sentences using the one-pass approach of previous summary report generators would be problematic. This is especially true

[9] The mention of the big board volume of the preceding day in the report of Fig. 6.1 is *not* historical information. It refers to the *initial* volume of the current day which coincides to the *final* volume of the preceding business day. The scope of this report is indeed limited to the current day.

[10] It is therefore by no mean the result of a rigorous, systematic comparison. It nonetheless gives a fair idea of the respective complexities of the sentences generated by these systems.

[11] There are of course many different possible ways to encode the content of a sentence in such predicate form. For the purpose of such comparison, the only concern is to remain consistent for sentences generated by different systems, which I have tried to do as much as possible.

for systems based on the Meaning-Text Theory (MTT) which are microcoded and lexically-driven. These complex sentences contain too many words to be assembled in one pass, and, as noted in section 2.4.1, their top-level structure is devoided of lexical constraints.

Scalability Among previous summary report generators, ANA remains the one generating the most complex and fluent sentences. It achieves fluency for fairly complex sentences by relying on a phrasal lexicon. The entries in this lexicon are phrases, comprising up to 8 words, simultaneously realizing several facts in an idiomatic way. To illustrate how this approach would generate STREAK's domain sublanguage, consider the final draft in the first example run of STREAK, given in section 4.4:
"Dallas, TX – Charles Barkley matched his season record with 42 points and Danny Ainge came off the bench to add 21 Friday night as the Phoenix Suns handed the Dallas Mavericks their franchise worst 27th defeat in a row at home 123 - 97."
Using a phrasal lexicon like the one of ANA, such a sentence could be *macro*coded from only 6 stored phrasal entries. In constrast, STREAK *micro*codes this sentence from 31 different entries made of indvidual words or collocations.

The advantage of the *macro*coded approach it is that it circumvents the identification of the complex constraints that influence the generation of these phrases. It has two drawbacks. First, it cannot exploit summarization potential lying below the clause rank. For example, the addition of a complex historical fact by only one or two words implemented in STREAK (cf. the first example run of section 4.4) could not be done with a phrasal lexicon.

Second, it makes scaling up the paraphrasing power prohibitively costly, since it requires the hand-coding of a combinatorially explosive number of phrases. For example, suppose that a word-based lexicon contains an average of 10 synonyms per concept. With only 310 entries such a lexicon could generate 10^{31} paraphrases of the sentence above[12]. Attaining such paraphrasing power with a phrasal lexicon similar to that of ANA would require hand-coding $10^{31/6} = 146,780$ entries. In section 5.2.5, I present a quantitative evaluation of the respective extensibility of the macrocoded one-pass approach and the microcoded two-pass approach, based on corpus data.

This scalability problem of ANA also applies to SEMTEX since it also relies on a phrasal lexicon. It does not, however, applies to generators based on the Meaning-Text Theory which build sentences from the individual word entries of the *Encyclopedic Combinatorial Dictionary* (ECD) whose format is an integral part of the theory. With this approach, the main problem is not scalability but rather sentence complexity (as explained in the previous section).

Paraphrasing power The extensive set of revision operations implemented in its revision rule base combined with the wide coverage of its syntactic grammar endows STREAK with high syntactic paraphrasing power. This power is illustrated on *Run 4* and *Run 5* given in appendix C. Among previous systems, only ANA focused on paraphrasing power. STREAK improves over ANA in this respect in two different ways. First, by covering syntactic constructions not covered in ANA such as infinitive clauses, relative clauses, appositions and nominalizations (cf. [Kukich 1983] p.137). Second and more importantly, it is able to convey the same fact at a variety of linguistic ranks, including below the clause rank. For example, the same fact expressed by a clause in "*Majerle came off the bench to score 24 points*" can be alternatively conveyed by a single word in "*Reserve Majerle scored 24 point*". Such alternative paraphrases cannot generated by a macrocoded generator where sentence subjects and sentence predicates are hard-wired in the lexicon and realize mutually exclusive classes of facts. This limitation thus applies to both ANA and SEMTEX.

The Meaning-Text Theory provides a comprehensive framework for handling paraphrasing. For example, the alternative between full-verb clauses and support-verb clauses which is captured in STREAK by the revision rules `nominalization` and `adjunctization` can be modeled in the MTT by the lexical functions $Oper_1$ (agent-oriented support verb) and S_0 (action nominal) (*e.g.,* S_0("to decrease") = "decrease" and $Oper_1$("decrease") = "to show"). However, it seems that this high paraphrasing potential of the MTT has not yet been fully exploited in the various implementations of the theory. Since there is only one realization

[12] Ignoring that some of these choices may be interdependent and hence somewhat reducing this total number.

Initial draft:
The one dimensional, zero based array of n elements, FLAG, represents the flag. There are three array markers, L, M, and R, standing for left, standing for middle and standing for right, respectively. L is initialized to 0. M is initialized to 0. R is initialized to n -1.
Final draft:
The flag is represented by a 1 dimensional, 0 based array of n elements, FLAG. There are three array markers, L, M, and R, standing for left, middle and right, respectively. L and M are initialized to 0. R is initialized to n -1.

Figure 6.7: Example text generated by YH: initial and final drafts (from [Gabriel 1988])

pattern per concept combinations in the target meteorological sublanguage of FOG (cf. [Polguere 1990] p.14-16), no paraphrasing power is implemented for that system. The paraphrasing power implemented in GOSSIP and LFS relies on lexical functions, but is limited to alternatives necessary to avoid repetitions within the same multi-clause sentences (cf. [Carcagno and Iordanskaja 1993] p.1021).

6.2 Related work in revision-based generation

In itself, the idea of revision in generation is not new. It has been proposed in different flavors by [Mann 1983], [Vaughan and McDonald 1986] [Meteer 1991], [Yazdani 1987], [Gabriel 1988], [Wong and Simmons 1988] [Cline and Nutter 1991] and [Inui *et al.* 1992]. However, only two implemented generation system emerged from these proposals: Gabriel's YH and Inui *et. al*'s WEIVER.

In what follows, I first briefly describe these two systems and compare the type of revisions they perform to those implemented in STREAK. I then review the non-implemented proposals put forward by the other authors cited above. The types of revisions described in these proposals differ from those implemented in STREAK in terms of: (1) the type of representation that is revised (at what level to revise?) and (2) the goals of the revision (why revise?). I compare revision in STREAK with previous proposals along these two dimensions in turn.

6.2.1 Yh

YH [Gabriel 1988] explores a number of interesting language generation issues on the toy domain of the Dutch national flag game. YH takes as input an abstract description of a LISP program playing this game and generates a paragraph describing the code. Since YH describes the *code* itself and not its execution (as in [Davey 1978] for example) and since YH does not include a module performing domain reasoning ([Gabriel 1988], p.29) the input to YH is the same for all its runs. Different outputs are obtained by playing with the value of adjustable parameters defining stylistic heuristics used during content organization and realization. YH is implemented in an object-oriented fashion. The expertise concerning various generation subtasks is procedurally encapsulated in the methods of different "objects".

YH is related to STREAK in that it works in two passes and performs both incremental generation[13] and revisions. However, it is during the initial draft building pass that YH works incrementally. In contrast, STREAK works incrementally during the subsequent draft revision pass. Moreover the type of revisions that YH performs during the second pass are *content-preserving*. STREAK performs *content-adding* revisions. For example, revisions in YH involve passivization, ellipsis and changing two conjoined clauses sharing the same verb and object, but each with a different subject, into a single clause with a conjoined nominal as subject. Such revisions do not make the draft more informative, but only more concise or coherent. An example initial draft generated by YH is given in Fig. 6.7. It is followed by the final draft once revised by YH.

[13] It therefore is related to the systems and proposal surveyed in section 6.3.

STREAK works incrementally starting from a draft representation that is already a complex structure. This complex structure is built prescriptively and guides the incremental revision process. In contrast, YH works incrementally from scratch and the utterance structure progressively emerges from locally planned and realized additions. Although providing for maximum flexibility, this type of incremental addition from scratch is underconstrained and may lead to the exploration of a very large search space. This expensive search is avoided in YH by relying on knowledge extremely specific to the tiny domain of the Dutch National Flag program. It could not be avoided in a more complex domain.

6.2.2 weiveR

WEIVER [Inui *et al.* 1992] generates paragraphs that provide information for library users, such as the library's schedule or the location of a particular book. It takes as input a semantic tree whose arcs are rhetorical relations indicating the high level organization of the text and whose nodes are the content units to convey. This semantic tree is thus similar to the SSS layer in STREAK, except that its structure does not yet reflect the low-level organization of content units inside phrasal constituents. To build a first draft, WEIVER traverses this semantic tree top-down, progressively replacing semantic subtrees by corresponding fully lexicalized syntactic trees. To perform this mapping, WEIVER relies on the sentence level content organization heuristics described in [Scott and Souza 1990] and a phrasal lexicon similar to the one described in [Jacobs 1985]. The final draft representation consists of a hybrid tree, whose top-levels are structured in terms of rhetorical relations and whose bottom-levels are structured in terms of syntactic dependencies. Each substitution of a semantic tree by a syntactic tree during the mapping above corresponds to one of the possible realization options encoded in the sentence planning heuristics and phrasal lexicon. Each syntactic tree is annotated with the semantic tree that is realizes and the option realization that it represents. After the full draft is built, each syntactic tree corresponding to a sentence is linearized.

The syntactic tree of each sentence is then examined by an evaluation component which computes stylistic parameters such as the lexical lengths and syntactic depths of various constituents. When this component detects a constituent that is too long or too deep, WEIVER backtracks to choose another sentence planning option, resulting in an alternative draft. WEIVER keeps track of the history of the options considered over multiple revision cycles. WEIVER also includes a facility for allowing a human editor to trigger similar backtracking when she detects an ambiguity in the latest draft generated by WEIVER. The issue of ambiguity is crucial in this work because WEIVER generates Japanese, a language where constituent ordering is very underconstrained by syntax which leads to numerous semantic ambiguities. This aspect of WEIVER underlines the interesting prospect of the revision-based approach for integrating language generation facilities in a comprehensive computer-assisted document production environement.

WEIVER differs fundamentally from STREAK in that all the revision operations it performs are *content-preserving*. For example, these operations include adding punctuation, re-ordering constitiuents or replacing an embedded clause by a separate sentence. None of these operations make the draft more informative: only less ambiguous and more readable. Also WEIVER does not attempt summarize the content it has to convey and is not constrained by space. Finally, WEIVER relies on a phrasal lexicon whereas STREAK uses a word-based lexicon.

6.2.3 At what level to perform revision?

Both Yazdani and Meteer (at least in her initial paper [Vaughan and McDonald 1986]) proposed performing revision from an actual natural language draft. At that level, revision is a three-step process: (1) rediscovering the generator's input anew by interpreting its output using a text-understanding system, (2) evaluating this output by matching its interpretation with the communicative goals that were input to the generator and (3) regeneration. The problem with this proposal is that the development of a text understanding system remains in itself such a tremendous endeavor that it cannot realistically be contemplated as a substep in the development of a revision component for generation, at least for generation-only applications like report generation. Using an existing natural language *understanding* system is not an option either because each of such systems works only for its very specific domain, sublanguage and application. Although some cross-domain robustness has been achieved by some *parsing* systems, their output is merely a partial parse tree annotated with some standard semantic features, which is far from the kind of thorough domain and goal-based *interpretation* needed to guide revision. This is why the other authors proposed, as I do, to perform revision directly on some representation of the draft internal to the generator. This is also perhaps why in her thesis [Meteer 1990], after having dismissed revision of an internal representation as mere "optimization" claiming "true revision" requires analyzing an actual natural language output, Meteer then proposes to perform revision on the Text-Structure, a level of representation internal to her SPOKESMAN generation system[14].

However, Meteer's distinction between "optimization" and "true revision" is pertinent because unless an internal representation has some special characteristics, revising it may end up being closer to backtracking and plan-elaboration during single-pass generation than it is to the draft reviewing and editing phase in two-pass generation. Of course, every form of revision *is indeed* a form of *global* backtracking while searching the N-dimensional space of all generation alternatives. It is important though to distinguish draft revision from *local* forms of backtracking restricted to a given class of generation choices. In my view, revising an internal representation is clearly distinct from backtracking and/or plan elaboration if that representation has the following characteristics: (1) it includes a layer "surfacey" enough to uniquely determine a natural language utterance and (2) it includes all unretracted intermediate decisions that ultimately led to that surface layer.

The first characteristic distinguishes draft revision from plan-elaboration because the latter is performed on a purely conceptual representation in one-to-many mapping with various natural language utterances. The second characteristic distinguishes draft revision from local backtracking because *the whole spectrum* of generation decisions is affected at once by a single revision[15]. The multi-layered FDs that STREAK uses to represent the current draft has both these characteristics. Its surface element, the Deep Grammatical Specification (DGS) uniquely determines a natural language output via the defaults provided by SURGE for unspecified input features. Moreover all its unretracted intermediate decisions are compiled in the Deep Semantic Specification (DSS), the Surface Semantic Specification (SSS) layers and the pointers maintaining the correspondence between two contiguous layers. These intermediate decisions are also maintained in YH inside its object system and in WEIVER inside its history of annotated hybrid trees.

Apart from distinguishing information-adding revision from plan-elaboration, the presence of a surface layer in the representation on which revision is performed is needed for another reason. As already noted in the previous section, summarization inherently involves trading off the conflicting goals of informativeness, conciseness and readability. By multiplying the number of candidate facts to convey, dealing with historical information only exacerbates the tension between these conflicting goals.

But factors like conciseness and readability directly depend on surface form and monitoring them cannot be done by reasoning only at higher layers. In that sense, Meteer's Text-Structure remains too abstract. Although grammatical constituency is already decided at that level, many grammatical features and open-class lexical items with different stylistic impacts are not yet specified. The level of representation that unequivocally determines a natural language utterance in SPOKESMAN is the Linguistic Specification input to MUMBLE-86 [Meteer *et al.* 1987]. Using SPOKESMAN levels of representation, revising to adjust the trade-off between informativity, conciseness and readability, would require acting upon both the Text-Structure level and the Linguistic Specification level.

Without being specific about the characteristics of the representation to revise, [Cline and Nutter 1991] also stressed the need for a representation to reflect both content planning and surface realization decisions. [Wong and Simmons 1988] advocate performing revision using a blackboard. Although they are not specific about what type of information is to be posted on the blackboard during revision, the inherent flexibility of such a mechanism would certainly allow manipulations to simultaneously affect several layers of decisions. In fact, the multi-layered FD representing the draft in STREAK can be viewed as a blackboard (cf. [Elhadad and Robin 1992] for a discussion of the special control mechanisms of FUF that render FDs blackboard-like).

[14] Note that Meteer's proposal is for future work. SPOKESMAN does *not* perform revision.

[15] In that sense, the hill-climbing phase of KDS [Mann and Moore 1981] is not revision but local backtracking.

6.2.4 Revising to satisfy what goals?

The revision goals considered by [Meteer 1991], [Gabriel 1988], [Wong and Simmons 1988] and [Inui *et al.* 1992] are purely stylistic. They are *information-preserving* revisions to make the draft more concise, clearer or more coherent. In contrast, STREAK performs *information-adding* revisions to make the draft more informative. [Cline and Nutter 1991] propose performing both varieties of revisions. However, the type of information-adding revision they propose (expanding during the revision of some unexpanded nodes of the textual schemata used for content planning during the first pass[16]) operates at the *discourse*-level (addition of a sentence or a paragraph) whereas the revisions implemented in STREAK operate at the *phrase*-level (addition of a clause, group or even a single word).

STREAK differs fundamentally from the the revision-based generators discussed so far in that it views as a way to gradually and opportunistically improve informative content of the report while keeping it short and concise. This view brings together revision-based generation with another line of research, incremental generation, reviewed in the next section.

6.3 Related work in incremental generation

Incremental generation has been investigated from three main perspectives: the AI perspective of YH already described in the previous section, the syntactic perspective of Tree Adjoining Grammars [Joshi *et al.* 1975] and the cognitive modelling perspective of IPF [De Smedt 1990]. I review research from these last two perspectives in the following sections.

6.3.1 Incremental generation and Tree Adjoining Grammars

Among the various approaches to incremental generation either proposed or implemented in recent years, a recurrent theme is the use of Tree Adjoining Grammars (TAGs) as an underlying syntactic formalism. TAGs have had a long and complex history in two adjacent research fields, formal language theory and natural language parsing, before people started to try to use them for natural language generation. In what follows I therefore first briefly overview the initial motivation of TAGs as well as their evolution from their inception to their use for generation. I then contrast the various TAG-based approaches to incremental generation with the approach presented in this thesis and implemented in the system STREAK.

6.3.1.1 Tree Adjoining Grammars

Introduced by Joshi, TAGs [Joshi *et al.* 1975] originated in the field of formal language theory, as an alternative to the various string rewrite formalisms, such as Context Free Grammars (CFGs) and Context Sensitive Grammars (CSGs), that had been put forward to study the computational complexity of both artificial and natural languages. The basic idea was twofold:

- Switch the building blocks for characterizing language from flat, unstructured strings to tree structures.
- Restrict the possible ways of combining these building blocks to a single operation called *adjoining*.

With this approach, the strings of language are no longer characterized, as in CFGs or CSGs, by a set of rewrite rules on a set of elementary strings. Instead, the strings of the language are indirectly characterized by a set of trees derivable through adjoining from a set of elementary trees. The strings are then available by the standard left-to-right reading of the tree leaves.

The adjoining operation involves merging two trees: an *initial* tree I containing a node of category X and an *auxiliary* A tree whose root is also of category X. Auxiliary trees have the special property of containing a leaf node whose category matches that of their root. This distinguished leaf node is called the *foot* node of

[16] Or in Hovy's term, expanding *growth-points* [Hovy 1990].

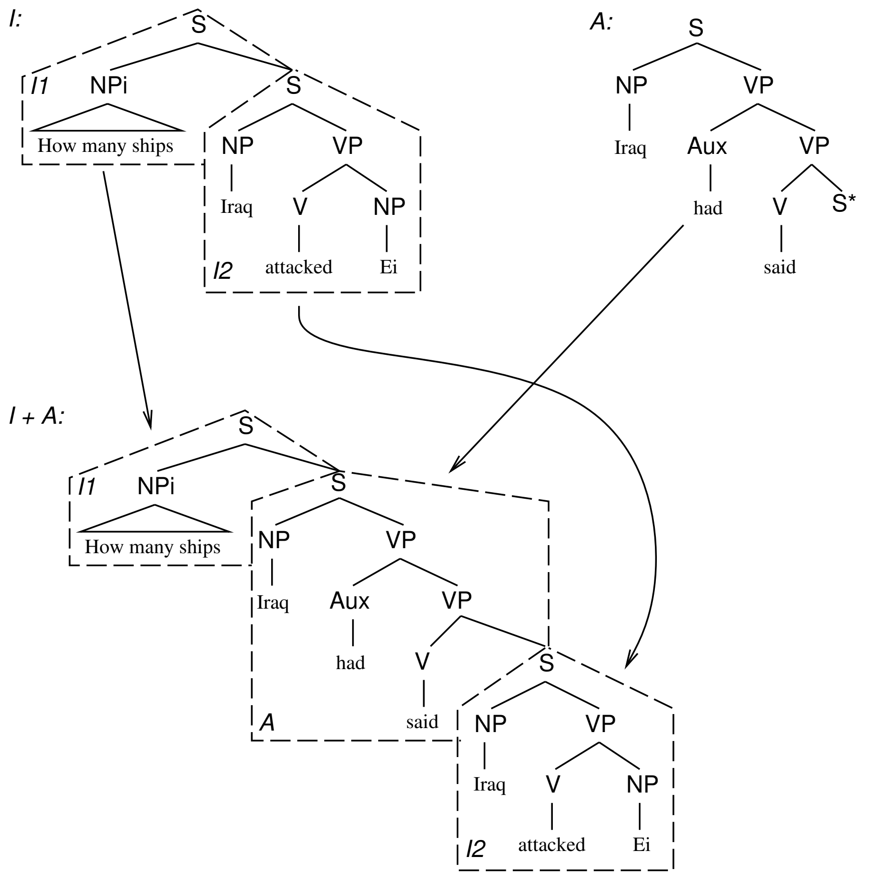

Figure 6.8: Adjoining in TAGs

the auxiliary tree. Ajoining A to I consists of excising the subtree I_x rooted at the node X of I, replacing it by A and then inserting I_x at the foot node of A. This adjoining operation is pictorially illustrated in Fig. 6.8. In this example, an indirect speech clause is adjoined to the original sentence:
"How many ships Iraq attacked?"
yielding: "How many ships Iraq had said that Iraq attacked?".

The auxiliary tree A is spliced in the initial tree I between the two subtrees I_1 and I_2.

Numerous studies of the formal and computational consequences of these two departures from previous grammatical formalisms were then carried out. It was shown that while being closer in terms of computational complexity to CFGs than to CSGs, TAGs can still model some of the phenomena of natural language syntax that are beyond the grasp of CFGs and previously required the much greater representational power of CSGs.

This result triggered a wave of enthusiam for TAGs in the natural language parsing community. However, researchers who attempted to use TAGs for parsing in the context of practical natural language applications, soon discovered that many of the syntactic phenomena encountered in the texts they attempted to parse, could not be satisfactorily modelled by the single operation of tree adjoining.

In response to this problem, TAGs then evolved through a long series of extensions, enriching the initial principle with various additions, many of them inspired from alternative, independently developed grammatical formalisms. Although the elegant simplicity of the original formalism based only on tree adjoining and now called "pure TAGs" has been lost along the way, many extensions motivated on practical grounds were then studied from the formal point of view. This resulted in an unprecedented wealth of theoretical

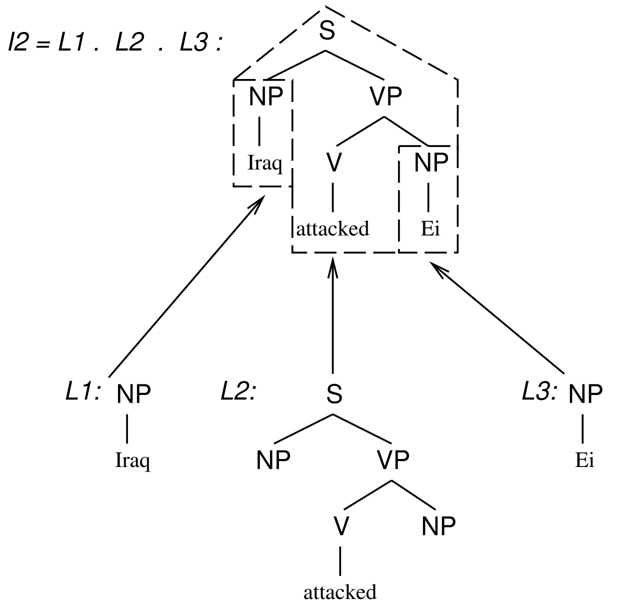

Figure 6.9: Substitution of minimal trees in lexicalized TAGs

results. Since many of these results concern extensions to TAGs inspired from other grammatical formalisms which had not previously been the object of such formal study, the evolution of TAGs created a synergic theoretical ground for the systematic comparison of diverse grammatical formalisms. Such comparisons bring much needed insights revealing fundamental and previously obscured differences and similarities among these formalisms.

The first notable extension to pure TAGs was the introduction of *Multi-Component* TAGs [Joshi 1987] that allow the simultaneous adjoining to the same tree of *sets of* auxiliary trees with co-referring nodes across several members of the set. Multi-component TAGs improve on pure TAGs in that they can model the long distance dependencies in sentences such as: "Which painting did you buy a copy of?".
The tree for this example sentence results from the simultaneous adjoining to the initial tree for the phrase "a did you buy?"
with the auxiliary trees for the wh-phrase "$which_i$ painting"
and the NP "a copy of e_i" with e_i co-referring to $which_i$.

The second and most dramatic extension of the formalism was the introduction of *Lexicalized* TAGs [Shabes *et al.* 1988]. In order to deal with lexical constraints on surface syntactic structure (*e.g.,* the type of complements a given verb accepts) and also to make the formalism more compositional, lexicalized TAGs depart from pure TAGs in two ways:

- The elementary trees are much smaller as they correspond to minimal structures representing the arguments of specific lexical items.
- The operation of *substitution* is introduced as a second way to combine elementary trees to supplement *adjoining*.

An auxiliary tree A with a root node of category X can only be substituted in trees with a leaf node of category X. The substitution itself simply consists of replacing this leaf node by A. Examples of substitions are given in Fig. 6.9. It shows how the tree I_2, which was a subtree of the pure TAG initial tree I in Fig. 6.8, is built from the three minimal lexical trees $L1$, $L2$ and $L3$ by two subtitution operations. This lexicalist import to TAGs seems very close in spirit to the surface syntax representation of the Meaning-Text Theory

[Mel'cuk and Pertsov 1987].

As a result of these two extensions, TAGs could now be used to compositionally derive most syntactic structures observed in natural languages. However, only lexical and syntactic constraints could influence the course of the derivation: the nodes of the trees were only labeled by a word and/or a syntactic category. The need to model semantic and pragmatic constraints as well, led to the introduction of *Functional* TAGs [Vijay-Shanker 1992], where each tree node is associated with a feature structure. The functional unification of these feature structures is used to constrain the adjoining and substitution operations. An auxiliary tree A can be adjoined (or substituted) to a node X of a tree T, only when the feature structure at the root of A unifies with the feature structure at X. The use of functional unification was inspired by Functional Unification Grammars (FUGs) [Kay 1979]. However, the resulting functional TAGs are even closer to Lexical Functional Grammars (LFGs) [Bresnan and Kaplan 1982], since they separate, on the one hand, dependency and precedence constraints represented by the trees, and on the other hand, the other types constraints represented by the feature structures, in very much the same way than LFGs. In LFGs dependency and precedence is captured by the c-structure and the other constraints by the f-structure. As explained in Appendix B FUGs represent dependency and precedence uniformly with all other type constraints inside the feature structures[17] respectively using the `cset` and `pattern` features (cf. Section B.1 of Appendix B).

The series of extensions above resulted from the desire to adapt a formalism initially designed as a theory of natural language competence to the needs of implementing grammars for parsing. A second series of extensions resulted from several attempts to adapt the formalism for the needs of implementing grammars for generation. This trend of work again originated from a paper by [McDonald and Pustejovsky 1985] in which they point out the similarity between the adjoining operation of TAGs and the *attachement* operation used in their system MUMBLE [Meteer *et al.* 1987]. Like SURGE used in this thesis and NIGEL [Mann and Matthiessen 1983], developped at ISI, MUMBLE is a portable syntactic processing front-end for the development of generation applications. It accepts as input a skeletal lexico-syntactic tree (called a *linguistic specfication*) and incorporates a fairly extensive portion of the English grammar. In constrast to SURGE which encodes the syntactic grammar declaratively as an FG, MUMBLE encodes it procedurally as a set of LISP functions[18]. NIGEL does a little bit of both, encoding the functional aspects of the grammar declaratively in a *system network* and its structural aspects procedurally as LISP functions called *realization statements*.

The interest of TAGs for generation continued with a paper by [Joshi 1987] comparing TAGs and CFGs and arguing that two properties of TAGs, extended domain of locality and flexible word-ordering specification, make TAGs more suitable to generation than CFGs. Four things must be noted concerning this comparison:

- It does not present an implemented TAG grammar for generation but rather contrast the two formalisms in terms of their general characteristics on illustrative example sentences.
- CFGs have been rarely used for the development of natural language generation systems (FUGs remains the most widely used formalism in generation). An exception to this general trend, may be the area of *reversible* grammars [Strzalkowski 1994]. Researchers in this field attempt to circumscribe a minimal set of modifications and extensions allowing the computational grammars and formalisms they developed for parsing to be re-used for generation as well. The hope is that for applications such as question/answering systems or interlingual machine translation where both a parser and a generator are needed, a reversible grammar would avoid duplicating the effort of writing the grammar rules twice (once for each module)[19].
- As pointed out in the paper itself ([Joshi 1987], p.243), the extended domain of locality of TAGs is also a property of FUGs. The flexible word-ordering specification property of TAGs (possible only for TAGs of the LD/LP variety discussed below) is also shared with FUGs (at least in the FUF implementation

[17] The particular feature structures of FUGs are called Functional Descriptions (FDs).

[18] With all the well-known software engineering drawbacks of procedual representations.

[19] This agenda seems to be strongly motivated by a desire of uniformity and formal elegance. From a purely practical, engineering viewpoint, the use of available stand-alone syntactic grammars for generation such as SURGE or NIGEL, whose coverage is already extensive and gradually being extended, seems a cost-efficient alternative for the development of such systems.

of FUGs, using paths in the `pattern` meta-feature as explained in Section B.2.1 of Appendix B). It must be noted however, that the limited power of TAGs (at least of pure TAGs) with its the desirable consequences in terms of computational complexity is *not* shared by FUGs.

- Joshi is not explicit concerning what variety of TAGs is compared with CFGs, but since he mentions only *adjoining* as way of combining elementary trees, it seems to describe pure TAGs rather than any of the extensions presented above that rendered TAGs practically usable for parsing. Without the operation of substitution that allows the derivation of syntactic structures from individual words, using TAGs for generation would require the use of a phrasal lexicon such as those of ANA [Kukich 1983], PHRED [Jacobs 1985] or PAULINE [Hovy 1988], with the limitations of this approach with respect to scalability discussed in section 6.1.4[20].

Considering this last remark it comes as no surprise that all subsequent proposals to adapt TAGs to the specific needs of generation used lexicalized TAGs as their starting point. Within this line of research, three main efforts have been:

- *LD/LP* TAGs [Harbusch 1994]
- *Synchronous* TAGs [Shieber and Shabes 1991].
- *Systemic* TAGs [Yang *et al.* 1991] [McCoy *et al.* 1992].

LD/LP TAGs (Local Dependence / Linear Precedence Tree Adjoining Grammars) depart from the regular lexicalized TAGs used for parsing in that they separate the expression of dependency constraints among constituents from the expression of precedence constraints among these constituents. These two types of constraints are implicitly compiled in the syntactic trees that are the building blocks of regular TAGs. For example the constraints encoded by the tree $L2$ in of Fig. 6.8 can be alternatively modularly encoded as follows:
dominate(S,NP1)
dominate(S,VP)
dominate(VP,V)
dominate(VP,NP)
dominate(V,attacked)
precede(NP,VP)
precede(V,NP)

This departure from lexicalized TAGs can be viewed as the logical continuation of the departure from pure TAGs to lexicalized TAGs in the evolution of the formalism towards more compositionality. LD/LP TAGs seems to constitute the final step in this evolution where the formalism has gone full circle to abandon its original distinctive characteristic: the use of syntactic trees as primitive elements. Indeed, in the way they separate the expression of dependency and precedence, LD/LP TAGs seem to stand closer to FUGs - where dependence is expressed using the `cset` feature and precedence using the `pattern` feature - than to other TAGs. This separation in expression allows LD/LP TAGs to decompose syntactic processing into two modules: one specifying a (possibly partial) syntactic structure and another linearizing this structure into a linguistic string. This decomposition parallels the division in SURGE between the grammar proper and the linearizer (cf. Section B.2.1 of Appendix B).

LD/LP TAGs have been used in the multi-media presentation system WIP [Andre *et al.* 1993] [Wahlster *et al.* 1993]. This system generates both graphics and natural language to provide explanations on how to operate an esspresso machine. In this type of application the emphasis is on the definition of the overall system architecture and on the task of coordinating the two media. The natural language generator is essentially used to produce locative sentences such as: *"The on/off switch is located in the upper left corner of the picture"*. In the comparable system COMET [McKeown *et al.* 1990] which generates coordinated textual

[20] For the example sentence of Fig. 6.8 above, the two phrasal patterns S_1 = "How many ships S Iraq attacked" and S_2 = "Iraq had said S" (where the S in each pattern indicating the node where adjoining can occur) would need to be stored as a whole in the lexicon.

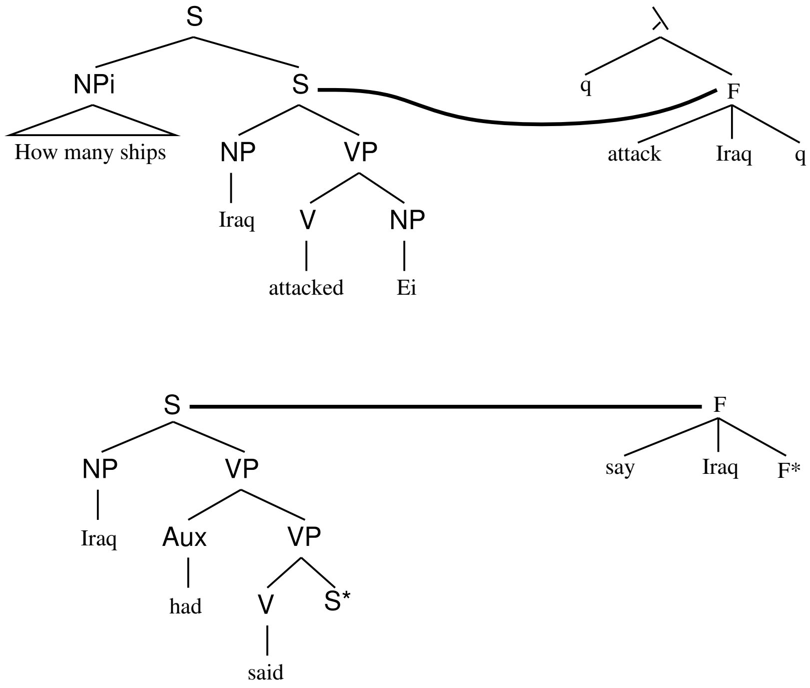

Figure 6.10: Mapping rules in synchronous TAGs

and graphical explanations on how to operate and repair a military field radio, similar locative sentences were simply generated in one-shot using the FUF/SURGE package.

One of the common characteristics of both pure TAGs and their parsing-motivated extensions is that they operate purely at the surface syntactic level. But a full-fledged generation system must start processing from an input that is a semantic as possible. The latest two extensions of TAGs attacked this deficiency of the formalism for generation.

In [Shieber and Shabes 1991] the authors propose to use for generation, the formalism of synchronous TAGs that they had developed for interpretation. A generator based on this extended version of TAGs would accept as input a logical form structured as a semantic tree. For example, the input for the sentence:
"How many ships did Iraq said were attacked."
would be: LF_1 = `say(Iraq,`λ`(quantity-of-ships,attack(Iraq,quantity-of-ships))`.

Note the non-isomorphism between the structure of that logical form and the syntactic structure of the corresponding sentence: the λ predicate which represents the main *wh*-clause, is embedded in the logical form and the `say` predicate, which heads the logical form, surfaces as a dependent clause in the sentence.

A synchronous TAGs consists of three sets of declarative structures:

- Elementary lexicalized syntactic trees with adjoining and substitution anchor points; they represent minimal linguistic constituents.
- Elementary semantic trees with adjoining and substitution anchor points; they represent minimal logical form constituents.
- Mapping rules associating a syntactic tree and a semantic tree and specifying the correspondence between their respective anchor points.

```
Input: ((speech-act wh-question)
        (wh-it {phenomenon actor})
        (tense past)
        (process "think")
        (actor "you")
        (phenomenon ((process "hit")
                     (tense past)
                     (actee "john")
                     (actor ((type person)
                             (id question))))))

Output: ``Who did you think hit John?''
```

Figure 6.11: Example input/output of a systemic TAG generator

Two example mapping rules are given in Fig. 6.10. Each associates a logical form on the right with a linguistic tree on the left. The thick link between the two indicates the correspondence between the anchor points where adjoining can occur on each side. The idea of synchronous TAGs is that, since these mapping rules are reversible, generation of the syntactic structure can be carried out by parsing the logical form using the semantic TAG. Each time a semantic tree is recognized, adjoined or substituted in the logical form being parsed, the mapping rule associated with the semantic tree is fired and as a side effect a corresponding syntactic tree is also chosen, adjoined or substituted. Generation of the linguistic output thus becomes a side effect of parsing the logical input. For example, parsing the tree corresponding to logical form LF_1 above would result in the generation of the syntactic tree $I + A$ in Fig. 6.8. Although very ingenious and appealing in theory, it remains to be seen whether this approach will turn out to be practical for the implementation of a generation application[21].

Systemic TAGs fully integrate the formalisms of systemic grammars and tree adjoining grammars. A generator based on a systemic TAG accepts as input a feature structure such as the one given in Fig. 6.11[22]. It is similar to the input accepted by SURGE in that it is structured in terms of thematic roles inspired from systemic linguistics. It differs in that it also specifies the closed-class words, that SURGE chooses on its own. The constituent structure of the input is traversed top-down, starting from the top-level clause and recursing on subconstituents. Each constituent is then passed through a systemic network encoding syntactic alternatives such as the various mapping of thematic roles into syntactic roles depending on constraints like dative or voice. The traversal of a subnetwork results in:

- The selection of an elementary TAG tree realizing the input constituent.
- The next subnetwork(s) to enter.
- A set of features to propagate to this next network(s).

The networks of systemic TAGs are thus exactly similar to those of NIGEL [Mann and Matthiessen 1983], a portable syntactic front-end for generation applications based solely on systemic grammars. The difference between the two lies in the way they build the syntactic structures satisfying the contraints accumulated during the systemic network traversal. For this task, NIGEL recourse to LISP procedures, whereas a systemic TAG uses the adjoining and substitution operations of TAGs. In that respect, systemic TAGs presents the advantage of being declarative instead of procedural. However, a generator based on systemic TAGs relies on an *heterogeneous* set of three declarative knowledge sources: the input feature structure, the systemic network and the TAG trees. It thus requires the implementation of three distinct interpreters, one for each

[21] Given the tricky control issues generally raised by the detailed procedural executions of reversible declarative models.
[22] Adapted to the FUF notation used for feature structures in this thesis.

source, plus a global control mechanism to control their interaction. This is in contrast with the simplicity and ecomony of a FUG-based generator such as FUF which relies only on the single operation of top-down recursive unification to perform the same task.

6.3.1.2 Contrast between STREAK and TAG-based approaches to incremental generation

In the previous section, I overviewed TAGs as a grammatical formalism and whenever relevant, contrasted them with FUGs, the grammatical formalism used in this thesis. I also surveyed the approaches to incremental generation proposed or implemented by the proponents of TAGs. In this section, I focus on contrasting these approaches with the approach to incremental generation implemented in STREAK. It differs from the TAG-based approaches with respect to three fundamental aspects:

- Monotonicity of the incremental generation process.
- Levels of processing incrementally performed.
- Emphasis of the research effort.

I elaborate on each of these aspects in what follows.

6.3.1.3 Monotonicity

The most important difference between STREAK and TAG-based approaches to incremental generation is that the latter are limited to *monotonic* operations on the sentence structure under construction. Both the adjoining and substitution operations of TAGs only further *elaborate* the sentence structure while preserving the expression of its original content. As opposed to STREAK, TAG-based generators never genuinely *revise* the sentence structure, altering the expression of its original content in order to accomodate a new fact. One of the most significant contribution of this thesis is precisely to show that relying only on elaboration only does not allow a generator to produce the most condensed linguistic forms needed in summarization applications such as newswire stories. The present thesis extends the scope of incremental generation research to encompass work on revision-based generation.

In spite of being hailed as especially suited for incremental generation, TAGs were deliberately designed to be monotonic (cf. [Shieber and Shabes 1991], p.226). They therefore do not present any advantage over other monotonic grammatical formalisms for the development of a non-monotic, opportunistic generator like STREAK. Implementing STREAK with TAGs would have required working out yet another extension of the formalism. The grammatical formalism chosen for the development of STREAK, FUGs, was also originally monotonic. The extension of the formalism to suit the special needs of non-monotonic applications is described in Section B.1.3.7 of Appendix B. The advantage of FUGs over TAGs for the development of STREAK lays primarily in the availability of the FUF/SURGE package. It allowed for a declarative and uniform implementation of all the components and provided an extensive initial syntactic coverage. It is the declarative nature of SURGE that rendered practical the extensions of its coverage to the target sub-language of STREAK[23]. In constrast, the procedural nature of MUMBLE - the only available TAG-based generation grammar of English with a significant initial coverage[24] - would have made such extensions hardly possible for an outsider.

In terms of specific transformations, the `adjoining` operation of TAGs corresponds to the following set of STREAK revision operations:

- The cases of `adjoin`[25], where a constituent of higher syntactic rank is adjoined to a constituent of strictly lower rank (*e.g.,* adjoin of relative clause to NP but *not* adjoin of temporal NP to clause).
- The cases of `absorb`, where the syntactic category of the absorbed and absorbing constituents match.

[23] And beyond, as described in Section B.3 of Appendix B.

[24] At the time I started the implementation of STREAK.

[25] Calling this transformation `adjoin` when it precisely corresponds a TAG operation that is *not* `adjoining` is admittedly confusing. However, I have to stick to this denomination for the sake of consistency with my previous publications.

- `conjoin` and `append`.

The `substitution` operation of lexicalized TAGs corresponds to the remaining cases of `adjoin` and `absorb`. No TAG operation corresponds to any of the non-monotonic revision operations of STREAK.

6.3.1.4 Levels of incremental processing

STREAK also differ from TAG-based approaches to incremental generation in that it handles three levels of processing incrementally: the deep semantic level, the surface semantic level and the deep syntactic level. The revision tools of STREAK simultaneously alter the DSS (deep semantic), SSS (surface semantic) and DGS (deep syntax) layers of sentence representation. Among TAG-based approaches to incremental generation the only one where transformations also simultaneously affect several layers of sentence representation is synchronous TAGs. Both the logical form, which represents the sentence at the surface semantic level and the TAG tree which represents it at the surface syntactic level are incrementally processed. Compared to STREAK it is thus more superficial and comprises only two layers. The three other approaches, MUMBLE, systemic TAGs and LD/LP TAGs seem to incrementally manipulate only the surface syntactic representation of the sentence.

6.3.1.5 Research emphasis

A third difference between the research presented in this thesis and related work on TAG-based incremental generation lies in their respective emphasis and orientation. Research revolving around TAGs tend to emphasizes formalism over data (the same few example sentences are found meticulously re-analyzed over mulitiple publications by the proponents of each new modification of the formalism). In constrast, with its extensive set of revision operations and its wide coverage of syntactic forms (in SURGE-2.0), the research presented in this thesis emphasizes data over formalism.

With it wealth of formal definitions, TAG-centered research is also theoretically oriented. New proposals tend to be evaluated for the elegance or computational properties of their formal characterization. In constrast, with its pervasive reliance on corpus data, the present research is empirically oriented. It evaluates the new concepts it proposes by implementing a practical generation application (summarization and contextualization of quantitative data) and by cross-examining corpus data across domains.

The research presented in this thesis is thus best viewed as bringing insights to the problem of incremental generation that nicely complement those of TAG-centered research.

6.3.2 Incremental generation in IPF

Aside from the TAG-based approaches, another line of research in incremental generation is due to [De Smedt and Kempen 1987] and [De Smedt 1990]. It is encapsulated in the system IPF (Incremental Parallel Formulator). IPF and STREAK investigate incremental generation topics that are almost entirely mutually exclusive. They aim at different types of output, model different levels of incremental processing, and encode different sets of transformations.

IPF is a simulation system whose goal is to provide an experimental testbed for cognitive models about the production of *spontaneous speech*, with special emphasis on errors and hesitations. This goal thus stands in sharp contrast with that of STREAK, which aims to model the production of *carefully planned and edited written texts*. This difference in output medium has consequences in terms of output complexity. In spontaneous speeches, sentences tend to be short since they are limited by the short-term memory of the speaker. Thus, IPF focuses on simulating the production of simple sentences such as "Otto eats an Apple" or "John and Mary seem to be at the party" (these are example of error-free sentences, examples simulating errors are given later on). In constrast, STREAK focuses on modelling the planning and editing of the lengthy written sentences used for summarizing and contextualizing complex events.

As input, IPF takes the equivalent of a skeletal lexicalized syntactic tree but specified procedurally as a LISP function. Except for its procedural encoding, this input thus contains the same information as the inputs to SURGE or MUMBLE. In STREAK, all the incremental processing concerns *building* such a skeletal lexicalized syntactic tree. Once built, this tree is then non-incrementally processed by SURGE. Thus, while STREAK works incrementally at the deep semantic, surface semantic and deep syntactic levels, IPF functions at the surface syntactic level. Among systems specializing at that level, IPF, MUMBLE and SURGE form a continuum. The sole goal of IPF is cognitive modelling, which can be done with its small grammar. In constrast, the sole goal of SURGE is portability across different practical generation applications, which requires a wide syntactic coverage. MUMBLE attempts to reconcile both goals within a single system. Within the grammar, IPF separates the expression of dependency and precedence constraints and is lexically driven. It thus seems very close to the LD/LP lexicalized TAGs used in WIP and discussed in the previous section.

In [De Smedt and Kempen 1987], the authors distinguish between six types of incremental additions to sentence structure during speech production[26]:

1. `upward expansion`, where the added fragment dominates the base fragment, *e.g.,* "John and Mary **are**".
2. `downward expansion`, where the base fragment dominates the added fragment, *e.g.,* "John and Mary are **at the party**".
3. `insertion`, where the added fragment splices the base fragment, *e.g.,* "John and Mary **seem to** be at the party".
4. `coordination`, where the base and added fragments are at the same structural level, *e.g.,* "John, Peter and Mary ... **and Anne**"
5. `reformulation`, where part of the base fragment is contradicted by the added fragment, *e.g.,* "You should have sent that letter ...uh... **handed it over**"
6. `lemma substitution`, where part of the base fragment is specialized by the added fragment, *e.g.,* "Do you really want to buy that record ...uh... **compact disc** ?"

There are two things to note about these incremental additions. First, the self-corrections contained in the last two types of additions are semantically non-monotic while remaining syntactically monotonic. Such a discrepency is peculiar to spontaneous speech and not available for written texts. None of the additions above are *both* semantically *and* syntactically non-monotonic as the most sophisticated revision operations implemented in STREAK.

Second, additions such as (1) allow both the input and output of the addition to be syntactically incomplete (hence the name fragment). This is also excluded for written texts. In STREAK,both the input and output of a revision are required to be complete, grammatical sentences.

Aside from these important differences, it is still possible to establish some correspondence between these additions and the operations used in STREAK and TAG-based approaches. `upward expansion` corresponds to some `absorb` cases in STREAK and to `lowering attachment` in MUMBLE. `downward expansion` corresponds to `adjoin` in STREAK and to `substitution` in lexicalized TAGs. `insertion` corresponds to some other `absorb` cases in STREAK, to `splicing attachment` in MUMBLE and to `adjoining` in TAGs. Coordination correspond to `coordinative conjoin` in STREAK.

In conclusion, IPF brings insights to the problem of incremental generation that complement both those of STREAK and of TAG-centered research.

6.4 Related work in evaluation for generation

Previous work in evaluation methods, especially quantitative ones, for language generation is very scarce. The reasons for this scarcity were discussed in section 5.1. In most cases, different approaches are compared

[26] In each example, the fragment added by the increment is highlighted in bold and "..." indicates an hesitation.

only qualitatively on a few well-chosen examples. The output of a generation system is considered satisfying when it produce grammatical sentences that are judged semantically accurate for the restricted sublanguage of the application. Issues such as coverage, extensibility and portability are discussed only qualitatively and rarely in much detail. The two notable exceptions to this general trend are the dissertations of Kukich [Kukich 1983] and Elhadad [Elhadad 1993b].

Kukich dedicates a full chapter to evaluation. She defines and estimates several quantitative parameters. These parameters compare the knowledge structures abstracted during a single round of corpus analysis with the knowlege structures actually implemented in the generator ANA. They thus quantitatively measure the coverage of *one particular implementation* with respect to *one sample* of the target sublanguage. This is in sharp contrast with the coverage evaluation presented in this thesis which evaluates a generation *model* (the revision-based microcoded approach) and estimates the influence of sublanguage sample size on knowledge acquisition *in general*. Kukich also discusses in some detail the same-domain extensibility and portability of the one-pass macrocoded generation model that she proposes. However, this discussion remains only *qualitative*. In contrast, I carried out a *quantitative* evaluation of these two properties for the revision-based microcoded model that I propose.

Elhadad briefly discusses evaluation in the conclusion of his thesis. One of the three main contributions of his work is the development of the response generator for the question-answering system ADVISOR-II. The system assists students planning their schedule for a semester. One the many important novelties of ADVISOR-II is its ability to generate linguistic constructs that express a subjective evaluation of the objective content also conveyed in the response. These constructs, that Elhadad calls "evaluative expressions", include argumentative connectives (*e.g., "although"*), judgment determiners (*e.g., "lots of"*) and scalar adjectives (*e.g., "difficult"*). Elhadad quantitatively evaluates the importance of such expressions by measuring their frequency of occurrence in a corpus advising session transcripts. Such measurement is similar to the percentage of newswire summary sentences containing floating concepts and/or historical information given in Chapter 2 of the present thesis. As an evaluation, these frequencies concerns only the *issues* addressed by the ADVISOR-II, not the solution that the system brings to these issues. Elhadad evaluates the solution only in *qualitative* terms. This is in contrast with the present thesis where *both* the issues addressed and the solutions proposed are *quantitatively* evaluated.

Chapter 7

Conclusion

To conclude this thesis I first revisit the contributions it makes to several subfields in natural language generation. I then discuss the limitations of the present work and the future directions in which it could be extended.

7.1 Contributions

In Section 1.3, I introduced the contributions this thesis makes to the field of natural language generation. I now revisit these contributions, specifying which of the following five subareas of language generation they are part of: summarization applications, system architecture, revision and incremental processing, evaluation and portable syntactic front-ends.

7.1.1 Contributions to summary report generation

The major contribution of this thesis to summary report generation is that it is the first to address to the three following issues:

- Providing the historical background for the summarized event.
- Planning and realizing sentences as complex and informative as human writers.
- Performing linguistic summarization below the sentence rank.

STREAK is the first summary report generator to provide the historical background of the new facts it reports. It thus not only *summarizes* a particular event but also *contextualizes* it. This constitutes a significant step towards bridging the gap between computer-generated reports and their human-generated counterparts. Conveying historical information allows for a much broader coverage. A generator ignoring such information could at best cover 35% of the sentences from the corpus of human-written summaries I analyzed. Taking into account the historical context also allows for a smarter choice of the new facts to report, since the relevance of a new fact is largely dependent on its historical significance.

STREAK is also the first generator that deliberately attempts to pack as many facts as possible within a single sentence. As a result, STREAK generates sentences up to the maximum complexity level observed in human-written summaries (46 words long, 10 levels deep, conveying 12 facts). Previous work focused on the generation of paragraphs where each sentence was significantly simpler (not over about 34 words, 6 levels deep) and less informative (no more than 6 facts).

Finally STREAK is the first generator performing linguistic summarization at the group and clause ranks. In previous work, this issue was investigated only at the sentence rank. The combined abilities to generate more complex sentences and to convey facts by adding only a few words in well chosen constituents allows

STREAK to produce more concise summaries, conveying more information (in some cases twice as much) in the same limited space.

Another contribution of this thesis is the corpus-based approach underlying the development of STREAK. This step-by-step methodology can be followed for the development of report generators in any application domain. It facilitates the four most delicate phases in the development of a generation system:

- Architecture design (as explained in Chapter 3)
- Knowledge acquisition (as explained in Chapter 2)
- Evaluation and testing (as explained in Chapter 5)
- Definition of stylistic preferences

The last point, defining stylistic preferences to choose among the different valid expressions of each concept combination in a given application domain is one of the most elusive generation problem. Without the guidance of corpus data, all grammatically correct forms need to be considered and solid evidence supporting any preference criteria is generally lacking. The corpus-based approach offers a very practical answer to this problem: consider only corpus occurring forms and choose between them in a semi-random fashion mimicking their respective frequency of occurrence in the corpus.

7.1.2 Contributions to generation architecture

The contribution of this thesis to generation architecture is the design of a new text generation model where a first draft conveying obligatory content with little expressive variation is incrementally revised to incorporate supplementary content with much expressive variation. This new model improves on previously proposed ones in that:

- *It makes the generation process more flexible* by allowing the decisions of which supplementary content gets included in the generated text as well as where and how they are conveyed to be made under a combination of semantic, lexical, syntactic and stylistic constraints. In particular, it allows using linguistic factors to monitor content planning, a requirement when concisely expressing the content of a whole paragraph in a single sentence without letting that sentence grow so complex as to be unreadable. This added flexibility thus allows the generator to express itself with a fluency more comparable to that of human writers.
- *It makes the generation process more compositional* in two orthogonal ways: by building sentences incrementally through a set of revision cycles and by separating the syntagmatic aspect of content realization (*i.e.,* the organization of content units inside the sentence) from its paradigmatic aspect (*i.e.,* the lexicalization of the individual units). This added compositionality facilitates the acquisition and extension of the generator's knowledge sources.
- *It is more cognitively plausible* for the generation of complex written sentences (as explained in Section 3.2.2).

This new model was shown operational by the implementation of the system STREAK. It was also shown to be far more robust than the traditional macrocoded one-shot generation model. In particular, a quantitative comparison of these two models (cf. Section 5.2) revealed that, in the average, the revision-based model multiplies coverage by over 2.5 and improves extensibility by almost a third.

Another interesting aspect of this new generation model is that it combines a unique set of properties whose desirability had been independently advocated in previous work:

- It distinguishes between foreground content to convey obligatorily and background content to convey opportunistically [Rubinoff 1992]
- It realizes floating facts across different linguistic ranks [Elhadad 1993b].
- It can conflate the expression of several facts within a single word [Elhadad 1993b]

- It uses declarative knowledge sources [Nogier 1990] [Polguere 1990].
- It is stratificational and modular [Polguere 1990].
- It uses a uniform underlying formalism (functional descriptions) to represent utterances at all levels of abstraction and all linguistic ranks [Simonin 1985] [Dale 1992] [Elhadad 1993b].
- It interleaves planning and realization [Appelt 1985] [Hovy 1988].

7.1.3 Contributions to revision-based generation and incremental generation

The major contribution of this thesis to revision-based generation and incremental generation is the extensive set of revision rules to incrementally incorporate quantitative and historical data in a given draft expressing related information. These revision rules constitute a new type of linguistic knowledge integrating semantic, lexical, syntactic and stylistic constraints. It is the first set of linguistic resources specifically geared towards incremental language generation to be based on a systematic, fine-grained corpus analysis. These revision rules were classified in a hierarchy of 141 classes and sub-classes. Their operationality was demonstrated by their implementation as a functional unification grammar in STREAK. Their relevance to quantitative domains other than the sports domain in which they were acquired was demonstrated by the fact that examples of their use were found in the financial domain for all 9 top-level classes and for a majority of the 53 bottom-level classes.

The other contribution that this thesis make to these two lines of research is to bring them together for the first time. Previous work on revision-based generation was limited to content-preserving transformations and previous work on incremental generation which was limited to monotonic draft elaborations. This thesis shows that in order to concisely accommodate a new content unit onto a draft sentence it is at times necessary to reformulate the original content of this draft through revisions.

7.1.4 Contributions to evaluation in generation

Another major contribution of this thesis is that it addresses the issue of evaluation in generation to a far greater extent than any previous work. In particular the three evaluations described in this thesis constitute the first attempt to:

- Quantify how much of a given sublanguage can be captured by various knowledge structures used for language generation and acquired by analyzing a sample of this sublanguage. It shows that in the average almost 80% of a year sample of the sublanguage can be covered by the revision rules acquired using a sample from another year.
- Quantitatively comparing different generation models and measuring the gain of modularity. It shows that revision-based generation is about 2.5 times more robust and a third more extensible than one-shot generation.
- Quantitatively assessing the degree of domain-dependence of knowledge structures used for generation and acquired from texts in a single domain. It shows that about 2/3 of the revision rules observed in sport reports are also used in stock market reports.

7.1.5 Contributions to SURGE

The contributions of this thesis to the portable syntactic generation front-end SURGE is to have extended its wide coverage at two essential linguistic ranks: nominals and complex sentences. At the complex sentence rank, I have added 78 <semantic-role,syntactic-category> realization pairs to the 16 original pairs of the adverbial sub-grammar. This extension resulted both from the addition of 22 new semantic roles and from a better coverage of the realization options for the 10 original semantic roles. At the nominal rank, I have added 10 realization pairs to the 20 that SURGE-1.0 already covered.

Since SURGE-1.0 already had a wide coverage at the determiner and simple clause ranks, SURGE-2.0 thus has a wide coverage at all four main linguistic ranks. This extended coverage combined with the ease of use that it derives from the uniformity, bi-directionality and declarativity of the underlying functional unification formalism makes it the best portable syntactic processing front-end for the development of generation applications available today. Since its development, SURGE-2.0 has been used as the syntactic front-end in three generation applications other than STREAK.

7.2 Limitations and future work

In this section I survey the limitations and future agenda of the research presented in this thesis. I first discuss the limitations of the FUF/SURGE software, starting with the implementation related ones which affect the performance of STREAK, and continuing with the more design related ones which affect the versatility of the package independently of its particular usage for the development of STREAK. I then discuss the limitations of the current implementation of STREAK not related to its reliance on FUF. I then depart from the focus of this thesis and discuss in what directions it could be extended. These directions have essentially two different aspects: (1) addressing the whole range of language generation beyond the phrase level content planning and content realization subtasks on which this thesis focused and (2) carrying out further evaluations.

7.2.1 Limitations linked to the use of FUF

7.2.1.1 Speeding up the revision process

One of the main limitations of the current implementation of STREAK is that it is quite slow. This is especially true for the generation of sentences approaching the complexity limit observed in the corpus. *Example run 1* presented in Section 4.4.1, where STREAK incrementally generates a 43 word sentence of syntactic depth 8 through five revision steps, took, for example, 2hr 18min 37sec. All the run times discussed here were obtained with a compiled version of the LUCID COMMON LISP code of STREAK, including the FUF/SURGE package, and performed on a SPARC-10 SUN workstation. In what follows, I briefly discuss the main source of this inefficiency and what could be done about it. Fortunately, it turns out to largely result from an artifact of the current implementation of FUF, which was inconsequential for the monotonic computations for which it was originally intended, but which renders it very inefficient for the non-monotonic computations performed in STREAK. Several projects currently under way at Ben Gurion University of the Negev are exploring various strategies to re-implement FUF more efficiently. As we shall see, they should bring tremendous improvement to the run time of STREAK.

In *example run 1* mentioned above, the generation of the initial basic draft, before revision starts, took only 21sec. It is thus clearly the way *revision* is implemented that slows down the overall generation process. However, it is not so much the number of revisions involved that is a factor as it is the complexity of the sentences being revised. *Example run 4* commented in Section C.20 of appendix C, involving seven revisions completed in only 43min 12sec. The difference between *run 1* and *run 4* is that, in *run 1*, revisions where chained to generate an increasingly complex sentence, whereas in *run 4*, each revision was carried out independently on the same basic draft to demonstrate the paraphrasing power of the system.

STREAK is thus fast in generating draft sentences, yet slow to revise them, especially as they become more complex. This observation points to the FUF functions that were specially developed to perform the cut and paste operations needed for implementing non-monotonic revisions as the probable source of this inefficiency. These functions, presented in Section B.1.3.7 of Appendix B, are `insert-fd` which pastes a smaller FD inside a larger FD as a sub-FD under a given path and `relocate` which conversely cuts as a self-contained smaller FD, a sub-FD under a given path inside a larger FD. I measured the respective proportion of total run time spent in these two functions on *example run 1*. The amount of time spent in calls to `insert-fd` was negligible. In contrast, the amount of time spent in calls to `relocate` was tremendous: it accounted for 83% of the total run time. Without these calls this total run time could be brought down under 23min.

When and why is `relocate` called during the course of the incremental revision process and could it be

```
FD0 = ((b {a})
       (d ((e {a c})))
       (a ((c 1))))

FD1 = ((b ((c 1)))
       (d ((e {b c})))
       (a {b}))

FD2 = ((b {a})
       (a ((c 1)))
       (d ((e {b c}))))
```

Figure 7.1: Canonic and non-canonic list representations of the same FD

avoided? To answer these questions, it is necessary to recall that the building blocks of the revision process are the following revision actions (introduced in Section 4.3.3.1):

- `add-fd` which pastes, using `insert-fd`, an additional feature (whose value is not a path) inside the revision workspace.
- `add-path` which pastes, using `fu`, an additional equation between two features inside the revision workspace.
- `del-fd` which removes, using `top-gdpp`, a feature inside the revision workspace.
- `cp-fd` which pastes in the revision workspace a copy of another feature in that workspace; the copy is obtained using `relocate` and the pasting is done using `insert-fd`.
- `map-fd` which calls the phrase planner or lexicalizer on a copy of a given feature in the revision workspace and then pastes the result of the call to another location in the workspace; the copy is obtained using `relocate`, the phrase planner or lexicalizer maps this copy using `uni-fd` and the mapping result is pasted using `insert-fd`.

It is also necessary to recall that there are several possible list representations for a given FD. For example, the three lists in Fig. 7.1 represent the same FD. In contrast, an FD is uniquely represented by a graph. The graph corresponding to the lists of Fig. 7.1 is given in Fig. 7.2.

FD_0 in Fig. 7.1 is the *canonic* list representation: all its paths are direct and its non-path values are located under the shortest (or in case of a tie, the first in alphabetical order) path among the equated features. So for example the value `1` is placed under `{a c}` rather than under `{d e}` that also points to it in the graph. FD_1 is non-canonic because physical representatives are not all under the canonic path. For example, `1` is placed under `{b c}` instead of `{a c}` violating the alphabetical convention. FD_2 is non-canonic because it contains an indirect path: `{b c}` where the value of `b` is itself a pointer, `{a}`.

The version 5.3 of FUF that I used to implement STREAK represents FDs as lists. The revision actions listed above appear in the Right Hand Side of STREAK's revision rules. Before the revision rule interpreter applies any of these actions it is imperative that the list L_d representing the FD which encodes the current draft, be in canonic form. Only such form allows predicting by looking at the features in a path, whether the value down this path in L_d is itself a path or not. Without such knowledge it would be impossible during the writing of a revision rule to determine whether the appropriate action to apply under a given path should be an `add-fd` or an `add-path`. This difficulty is illustrated by the set of revision actions to build the canonic FD_0 from scratch:

```
(add-fd ((c 1)) {a})
```

```
(add-path {a} {b})
(add-path {a c} {d e})
```

These have nothing in common with the corresponding the set of revision actions to build the non-canonic FD_1 representing the same FD:

```
(add-path {b} {a})
(add-fd ((c 1)) {b})
(add-path {b c} {d e})
```

Though writing revision rules requires assuming that they will be applied to a canonic form FD, the version 5.3 of FUF in which STREAK was implemented, the unification operation does *not* preserve the canonic FDs being unified. This unfortunate property is illustrated by the following example run:

```
> (setf FD3 (fu FD0 '((f {a c}))))
((b {a})
 (d ((e {a c})))
 (a ((c 1)))
 (f {a c}))

> (setf FD4 (canonize FD3))
((b {a})
 (d ((e {f})))
 (a ((c {f})))
 (f 1))
```

In FD3, the physical represent of the class of equivalent paths: `({d e}, {a c}, {f})` is not located under the shortest path of the class, `{f}`, but instead under the longer `{a c}`. FD3 is thus non-canonic. The canonic list representation of the FD resulting from this unification operation is FD4. Since several revision actions, such as `add-path` and `map-fd` rely on unification, after each such action, the draft must be *re-canonized* before the application of the next action. This canonization is done by calling `relocate` (which is guaranteed to produce a canonic FD as explained in Section B.1.3.7 of Appendix B.) on the draft FD with a empty (*i.e.,* `{}`) relocation path.

During the application of a revision rule, `relocate` is thus called in two different types of circumstances:

1. To canonize the onion-layer FD representing the whole draft after the application of an action that may have altered its canonicity.

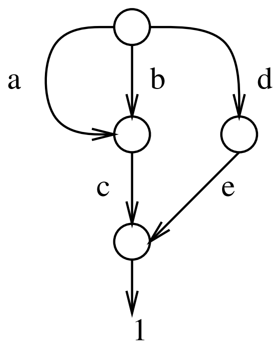

Figure 7.2: Unique graph corresponding to the lists of Fig. 7.1

2. To cut from the whole draft structure a sub-structure which constitutes the scope of a `cp-fd` or `map-fd` revision action.

In *example run 1*, 58% of the overall run time is spent in relocations called for canonization purposes and 24% in relocations called for the purpose of cutting sub-FDs. Skaliar [Skaliar 1994] describes an on-going effort to implement a new version of FUF which internally represents FDs directly as quotient sets as opposed to lists as in the current version. Such graph-based version would prevent the need for re-canonization, thus already more than halving the worst-case run time of STREAK. Furthermore, as explained in Section B.1.3.7 of Appendix B, `relocate` consists of three internal steps:

1. Convert the larger FD to quotient set form.
2. Compute the quotient set of the sub-FD under the cutting scope.
3. Convert back the sub-FD quotient set into list form.

With a graph-based version, since all FDs are already in quotient set form, only the second step is required. Since this second step is of negligible duration compared to the two others, the remaining run time spent for relocation with the current list-based version would also virtually disappear. Using such a version, the run time for the most complex sentences would thus fall down to under 25min.

This last run time would be for an *interpreted* graph-based version. Two other projects currently at their inception, involve the development of two *compilers* for FUF programs: one whose object code is compiled COMMON-LISP code and another whose object code is compiled C code. The compiled LISP version is expected (cf. Elhadad, personal communication) to reduce the average run time of a typical FUF program by 10 and the compiled C version by 20 . With these expected optimizations, the run time of STREAK to generate the most complex sentences observed in the corpus of human-written summaries would fall to the order of the minute.

Note that for the generation of written reports much slower run time is acceptable in the context of practical applications for which the architecture of STREAK was designed. Written reports can be generated off-line. Even though there is some time pressure involved in generating texts such as newswire stories, it is in no way comparable to the pressure involved in other generation applications such as the production of responses in an interactive dialog system.

7.2.1.2 Negation and variable attributes

The revision rules of STREAK are, as they stand, already very generic. This genericalness is achieved by two properties of STREAK's architecture. The first property is the top-down traversal of the draft constituent structure built in the revision rule interpreter. This traversal allows the revision rules to be local to a given type of constituent, independently of where this constituent appears in the particular draft structure at hand. The second property is the `map-fd` action that allows revision rules to specify only where and how to attach the new constituent expressing the content unit added to the draft by the revision, leaving the responsibility of specifying the internal structure and linguistic form of this new constituent to phrase planning and lexicalization rules.

Two aspects of FUF as the underlying programming language of STREAK prevent the expression of even more generic revision rules. The first aspect is the absence of an elegant way to express negative constraints in FUF and consequently in the LHS of STREAK's revision rules. The only way to encode such constraints in the current implementation is to rely on the procedural meta-attribute `control`, as in the example LHS below:

```
;; Additional game statistic by a different player
;; than the one with which it is coordinated
((lhs ((CONTROL (not (eq (get-adss-carrier ,n)
                         (get-conjoined-stat-carrier))))))
```

Had FUF a meta-attribute to express negation, the constraint above could be expressed declaratively by a path inequality instead of procedurally via the calls to the `get-adss-carrier` and `get-conjoined-stat-carrier` functions.

Another aspect of FUF that limits the genericalness of STREAK's revision rules is the lack of either wild-cards in paths or of a facility to define a hierarchy over attributes similar to the `define-feature-type` construct for the definition of a hierarchy over atomic values.

Consider the following example disjunction:

```
(alt (((lhs ((bls ((partic ((range GIVEN)))))))
       (rhs ((cp-fd {bls partic range} {pred-modif instrument np}))))
      ((lhs ((bls ((partic ((created GIVEN)))))))
       (rhs ((cp-fd {bls partic created} {pred-modif instrument np}))))
      ((lhs ((bls ((partic ((affected GIVEN)))))))
       (rhs ((cp-fd {bls partic affected} {pred-modif instrument np}))))))
```

which encodes one action of the various specializations of the revision rule *adjunctization of object as instrument* in terms of the thematic role filled by the object being adjunctized. The same constraint could be more generically expressed without this alternation by defining the roles `range, created` and `affected` as specialization of a generic `object` role for example as follows:

```
(define-attribute-type object (range created affected))

((lhs ((bls ((partic ((object GIVEN)))))))
 (rhs ((cp-fd {bls partic object} {pred-modif instrument np}))))
```

Unfortunately the definition of an attribute hierarchy would require fundamentally rethinking the semantics of the unification operation. For example, given a hierarchy defined as
`(define-attribute-type a (a1 a2))`
what should be the result of the unification of `((a1 1) (a2 2))` with `((a 3))`?
Should it be `((a1 1) (a2 2) (a 3))`?, `:fail`?, something else? A careful examination of such fundamental semantic questions should be carried out before any attempt to implement this particular extension of the formalism. It thus raises difficult design issues. In contrast, the addition to FUF of such meta-attribute for negation seems a simple implementation matter.

7.2.2 Limitations proper to STREAK

7.2.2.1 Extending the ontology of quantitative domains

The sports domain ontology covered by STREAK is restricted to three main classes of information:

- Box score statistics: data summarizing the various aspects of the overall performance of a player or a team during the reported game (*e.g.,* "Dennis Rodman grabbed 24 rebounds").
- Historical records: the property of one such statistic to constitute a record over some period of time extending at least across several games (*e.g.,* "Shaquille O'Neal scored **a season-high** 45 points").
- Historical streaks: the property of a performance to extend or interrupt a series of similar performances by the same team or player (*e.g.,* "the Golden State Warriors defeated the Charlotte Hornets 109-102 **for their seventh straight win**.").

As explained in Section 2.2, these three classes were chosen for two reasons. First, they are not peculiar to sports. Any quantitative domain has content classes which closely parallels them. For example, final game statistics in the sports domain directly correspond to final stock index values in the financial domain and streaks of victory/defeat for a sports team directly correspond to streaks of raise/decline for a financial index. Second, they are the most frequently reported facts in the corpus of human-written summaries that underly the development of STREAK: in 50% of the corpus sentences box score statistics were the only type of non-historical information and in 62% of them, records and streaks were the only type of historical information.

It would be interesting to extend the ontology of STREAK to less frequent types of both historical and non-historical information, which also have the property of being pervasive in multiple quantitative domains[1]. Notably, the extended ontology could cover the following additional classes:

- Non-historical records: similar to historical records but within to the temporal scope of the reported event (*e.g.,* "Hakeem Olajuwon scored **a game high** 37 points"
- Non-historical streaks: similar to historical streaks but within to the temporal scope of the reported event (*e.g.,* "Sam Cassell scored **all** 12 **final** points for Houston").
- Near records: statistic whose value comes close to equalling a record (*e.g.,* "Olajuwon added 13 assists, **one shy of his career best**").
- Near streaks: set of similar events who quasi-totality are of the same type "the San Antonio Spurs defeated the Utah Jazz 103-97 **for their 12 victories in the last 14 games**"
- Historical counts: absolute number of performance above a given statistical threshold (*e.g.,* "Scottie Pippen recorded **his eighth career** triple-double, with 19 points, 10 rebounds and 11 assists").

7.2.2.2 Developing global backtracking facilities

STREAK consists of four independent modules: the phrase planner, the lexicalizer, the syntactic grammar and the reviser. In the first three, the internal control mechanism is the top-down recursive functional unification built in FUF. The control mechanism of the reviser is more complex. First, a revision cycle is divided into two stages: the search for an applicable rule, followed by the application of the found rule (when the search was successful). The outer control loop of the first stage consists of a top-down traversal of the draft structure. The inner control loop for each subconstituent in this structure consists of the *non*-recursive functional unification (also built in FUF) of the revision rule base with the pair:
< current draft , additional content unit >.

The control mechanism of the second stage is the simple traversal of the revision action list associated with the rule found during the first stage. Some elements in this list can be a `map-fd` action, which triggers a call to either the phrase planner or the lexicalizer in the current revision context. The overall control mechanism for the generation of a sentence is quite complex: it consists of an initial drafting pass where phrase planner and lexicalizer are called in turn, followed by a revision pass made of a number of revision cycles.

The `u-exhaust` FUF function can be used to trigger systematic automatic backtracking separately in the phrase planner, lexicalizer and syntactic grammar. However, the current implementation does not include a facility for systematic automatic backtracking, neither for the reviser nor for the overall generation system. During a given revision cycle, STREAK always applies the first applicable rule it finds and moves on to the next cycle. A useful extension of the implementation would be to develop facilities to backtrack:

- Within a single revision cycle.
- Over a whole revision pass made of several revision cycles.
- Over the whole generation process (encompassing both the draft and revision passes) from a given pair < Fixed facts DSS, stack of floating fact DSSs >

A facility to backtrack over one revision cycle would allow exhaustively demonstrating and testing the revision-based paraphrasing power encoded in the system. It would allow for example, generating the complete list of revised drafts possible from a given pair < additional content unit , current draft >. Maintaining

[1] Though possibly necessary for a real-world application, extending the ontology to less frequent content classes peculiar to sports would not, from a purely research standpoint, present any particular interest.

the current point in the search space explored by such a backtracking facility is far from trivial. This is because there are four distinct sources of non-determinism at play during a revision cycle:

- The various draft subconstituents onto which the new constituent can be attached.
- The various revision rules that can be used to perform this attachment on a given subconstituent.
- The various phrase planning rules that can be used to define the internal structure of this new constituent.
- The various lexicalization rules that can be used to realize each element in that internal structure.

A slightly modified version of the `u-exhaust` FUF function could serve as the building block for the implementation of such a backtracking facility in STREAK.

A facility to backtrack over a whole revision pass encompassing several revision cycles would ensure that the maximum number of floating facts incorporable within a given draft gets included in the final summary. With the current implementation this is not guaranteed. When several revision rules are applicable during revision cycle N, the first one found by STREAK may not be the one whose application would result in the most concise revised draft. It is nonetheless applied. It then becomes possible that, during cycle N+M, only one revision rule is applicable but it would result in a revised draft beyond the maximum complexity observed in the corpus of human-written summaries. In this case, STREAK will not be able to incorporate this last floating fact due to lack of space. However, it is conceivable that had an alternative revision rule resulting in a more concise form be applied at cycle N, the complexity threshold would not be reached at cycle N+M. With a backtracking facility across several revision cycles, it would be possible to undo a choice of a revision rule at a given cycle and resume the revision process from this point in an attempt to include more floating facts.

Finally, a system-wide backtracking facility covering both the drafting and the revision passes would allow exhaustively demonstrating and testing the overall paraphrasing power encoded in the system. This power is very high due to the multiplicity of paraphrasing sources at work during the generation of a given sentence. There are two such sources at play at draft time: alternative phrase planning rules and alternative lexicalization rules. And there are four sources at play at each revision cycle: alternative draft constituents on which to perform the revision, alternative revision rules, and again alternative phrase planning rules and lexicalization rules for realizing the floating fact in the context of the revision. For example, suppose that for a given pair, $<$ Fixed facts DSS , Stack of floating fact DSSs $>$, there are only two alternative phrase planning rules and lexicalization rules available both at draft time and at each revision cycle. Further suppose that, at each cycle, there are also only two alternative draft constituent on which to apply only two alternative revision rules. Even in that case, a sentence generable in 5 revision cycles would nonetheless have $2 \times 2 \times ((2 \times 2 \times 2 \times 2)^5) = 4,194,304$ paraphrases. That so many forms can be generated from the same semantic representation with only $2 + 2 + ((2 + 2 + 2 + 2) \times 5) = 44$ rules attests to the extreme compositionality of the generation approach implemented in STREAK.

7.2.3 Implementing fact generation and discourse planning

As explained in Section 3.1, the overall text generation task comprises the following subtasks:

1. Content production (retrieving all the facts to potentially report)
2. Content pre-selection (choosing a restricted set of candidate facts)
3. Content final selection (picking the candidate facts to be included in the summary)
4. Discourse level content organization (assigning facts to sentential slots in the summary)
5. Phrase level content organization (assigning facts to sentence constituents)
6. Lexicalization (choosing the open-class words of each sentence)
7. Semantic grammaticalization (choosing the syntactic structure of each sentence)

8. Morpho-syntactic grammaticalization (applying grammar rules, inflecting open-class words and choosing closed-class words)
9. Linearization (spelling out the inflected words in order)

In chapter 3, I proposed (in Section 3.3.2) a complete generation architecture for a system performing all these tasks. However, in the rest of the thesis I focused on tasks 3 and 5 to 9. The current version of STREAK implements only these tasks. In the following subsections, I briefly discuss the interesting issues raised by extending this implementation to tasks 1, 2 and 4 as well.

7.2.3.1 Starting from the raw numbers

Currently, STREAK accepts as input two sets of conceptual networks respectively representing the fixed facts to convey obligatorily and the floating facts to convey opportunistically. Transforming STREAK from the research prototype that it currently is into a complete generator summarizing and contextualizing basketball games directly from raw quantitative data would require implementing an additional module in the architecture proposed in Section 3.3.2: the fact generator. The input to this fact generator is a box-score containing the final statistics of the game. In addition, this module has to have access to a historical database about basketball. The generation tasks to be implemented by this module are content production and content preselection. It would perform *conceptual* summarization, complementing the *linguistic* summarization already implemented in the current version.

It would probably be best decomposed into specialized components respectively responsible for:

1. Reading the input table and producing a record for each statistic.
2. Querying the database for related historical statistics.
3. Reformatting each statistic in the symbolic form that the phrase planner and reviser need as input.
4. Discriminating between fixed and floating facts.
5. Computing the relevance grade of each floating statistic in the input table and each historical fact related to it.

The task of the first three is fairly straightforward and would probably be best implemented as C programs. In contrast, the task of the last two involve encoding domain expertise and would be best implemented declaratively, possibly as FUF programs.

From a research perspective, the implementation of these components does not seem to raise any especially challenging issue, except perhaps the assignment of relevance grades. However, the availability of such module in STREAK would allow the carry out an ambitious systematic evaluation of the overall implemented system by running it on the box-scores of the new games played everyday.

7.2.3.2 Multi-sentential revision-based generation

Currently, STREAK summarizes its input data by a single complex sentence. It generates the type of *lead* sentence observed in the corpus of human-written newswire reports. As explained in Section 2.2, such sentences were chosen as the focus of this thesis because they themselves summarize the rest of the reports. The most intriguing research direction to follow within the new revision-based framework put forward in this thesis is to aim at generating whole newswire reports made of multiple sentences. Extending STREAK to carry out such a task would require implementing the last module in the architecture proposed in Section 3.3.2: the discourse planner.

In itself, the implementation of such a discourse planner should not pose any fundamental difficulty. Since within the architecture proposed in this thesis, the discourse planner needs to handle only the organization of *fixed* facts – *floating* facts being handled by the reviser – the standard technique of textual schemas

[McKeown 1985] would be very appropriate. The limitation of this technique is precisely its difficulty to cope with *floating* facts.

Instead, the difficulty of moving from sentence generation to multi-sentential text generation with a draft and revision approach, lies in how this change affects the task of the reviser. It raises a set of new issues that I now review.

Adequately revising a multi-sentential draft would involve identifying and encoding *discursive* constraints on the maximum sentence complexity allowed for each sentential slot in the report structure. Even though it would allow for maximal conciseness, the strategy of generating a report consisting exclusively of very complex sentences would be stylistically inappropriate. In human-written summaries, complex sentences alternate with simpler ones (probably to avoid taxing the reader's concentration excessively). When they conflict, what is the best trade-off between informativeness and stylistic appropriateness?

In a multi-sentential setting the control mechanism of the revision process needs to traverse two lists (instead of one): the list of draft sentences and the list of floating facts. Which of these two traversals should be the outer loop of the reviser?

Though I have not verified in the corpus whether it is indeed the case or not, in a multi-sentential setting it becomes possible that some obligatory facts are floating (*i.e.,* they are found in every corpus report but in different sentential slots in different reports). Would these facts be best handled by the reviser or by the discourse planner?

It may also be the case that some sentential slots in the report structure have no fixed facts associated with them. What should constitute the anchor point for initiating the use of such sentential slots – empty in the first draft – to convey a floating fact?

Given the much larger search space of alternatives provided by a multi-sentential setting, how much and what type of backtracking should be allowed without rendering the overall generation process inefficient?

Currently, the only goal of the reviser is to make the draft more informative by incorporating new floating facts. As explained in Section 3.2.1 this is best done at revision time because constraints on such incorporations are both *non*-local and heterogeneous (semantic, syntactic, stylistic and discursive). The non-locality is not a problem for the reviser since it can search the entire draft structure for a constituent on which to apply a revision rule. The heterogeneity is also not a problem since all types of constraints are captured by the multi-layered draft representation that the reviser manipulates. Constraints on pronominalization and abridged forms of subsequent references are also non-local and heterogeneous. They could also be best handled by the reviser. In other words, default forms of reference could be systematically generated at draft time and then carefully abridged at revision time. This new agenda for the reviser raises the following question: should the respective application of reference abridging and information adding revision rules be separated or interleaved and how?

7.2.4 Further evaluations

One last interesting direction in which the research presented in this thesis could be pursued is to carry out further evaluations.

The evaluations presented in this thesis focused on quantitatively evaluating both the new generation *model* and the new linguistic *knowledge structures* onto which the STREAK system is based. However, the implementation itself was not directly quantitatively evaluated. Because in its current version STREAK accepts as input hand-coded symbolic inputs (as opposed to the raw number tables available on-line) and does not include a facility to trigger system-wide backtracking, it has only been tested on a representative yet limited set of inputs and not all possible outputs have been generated from each of them.

While STREAK's accuracy is largely insured by the fact that STREAK is based exclusively on constraints individually observed in the corpus, some constraint *combinations* which have not been probed by the current test set could possibly yield to poor wording or style. This is the unavoidable downside of the robustness gained through the use of declarative partial constraints compositionally combined through functional unification.

Interactions between lexical and stylistic factors may be especially tricky to unearth, as shown by the following constraint discovered with the existing test set for the nominal expression of a streak:

1. "extended their winning streak to 12 straight game"
2. "extended their winning streak to 12 straight $\emptyset$"
3. "extended their winning streak to 12 consecutive game"
4. **?** "extended their winning streak to 12 consecutive $\emptyset$"

Even though in the four sentences above the adjectives *"straight"* and *"consecutive"* are synonymous and occupy the same head noun pre-modifying syntactic function, *"straight"* can be used in conjunction with ellipsis of the head noun but *"consecutive"* cannot. Since unfelicitous forms such as sentence (4) above never appear in the corpus, foreseeing the idiosyncratic constraint interaction from which they result before testing is very difficult. Fortunately, while the declarative and compositional framework of functional grammars may lead to some over-generation problems due to such unforeseen constraint interactions, it also makes their correction easy once detected. In the example above, all that was needed was a single line of additional code testing the pre-modifying adjective when deciding whether to elide the head noun.

As noted earlier, a pre-requisite to systematic implementation testing is the development of the fact generator that would make STREAK a complete generation system going all the way from the raw statistics to the natural language text summarizing and contextualizing them. The same evaluation methodology based on acquisition *vs.* test corpora used in this thesis at the more abstract level of knowledge structures could be used at the more detailed level of actual system runs. For example, it would be interesting to systematically compare over a whole season of new reports the output that the system produces from the box-score for each game with the corresponding reports produced by human-writers.

Finally, consider again the portability evaluation of the revision rules used by STREAK presented in Section 5.3. It measured the degree of *domain* dependence of these rules. It would also be interesting to evaluate their degree of *language* dependence. This would require developing a multi-lingual version of the system. It is therefore a rather long term goal, if only because of the overhead involved in developing a syntactic grammar equivalent to SURGE in at least one other language than English. It could nonetheless bring important insights in terms of exactly what type of constraints should be encoded at each layer of the utterance representation internally used by the generator.

Appendix A

A Complete Inventory of Revision Operations

The corpus analysis presented in Chapter 2 resulted in a hierarchy of the revision operations. The top of this hierarchy - shown again in Fig. A.1 - distinguishes between *monotonic revision tools* which consist of attaching a new constituent to the draft while preserving the surface form of its original content and *non-monotonic revision tools* which involve modifying the expression of the original content in order to accommodate the additional content in the draft syntactic structure. The bottom distinctions in this hierarchy subdivide each class of revision *tools* into a set of revision *operations*. A revision operation is a revision tool possibly accompanied by a *side transformation* which corrects whatever information redundancy, ambiguity or invalid lexical collocation the revision tool application may have introduced.

In Chapter 2, only one class of monotonic revision tool, one class of non-monotonic revision tool and one side transformation were discussed in detail. In this appendix, I give a detailed presentation of all the revision tools and side transformations. Moreover, I present a more complete version of the revison operation hierarchy. The hierarchy outline in Section 2.5 results from the original corpus analysis covering only a single season of basketball reports. The hierarchy presented in this appendix also includes the new revision operations discovered during the analysis of two additional seasons of basketball reports (which was performed for coverage evaluation purposes as explained in Section 5.2). Finally, while the revision examples in Chapter 2 come only from the basketball domain, this appendix also gives example from the stock market domain (which as used for the portability evaluation presented in Section 5.3.

For each class of revision tool in fig. A.1, I provide the following:

- Its general revision schema.
- How it specializes into subclasses down the revision operation hierarchy (the number of corpus occurrences of the most specific classes are given below them).

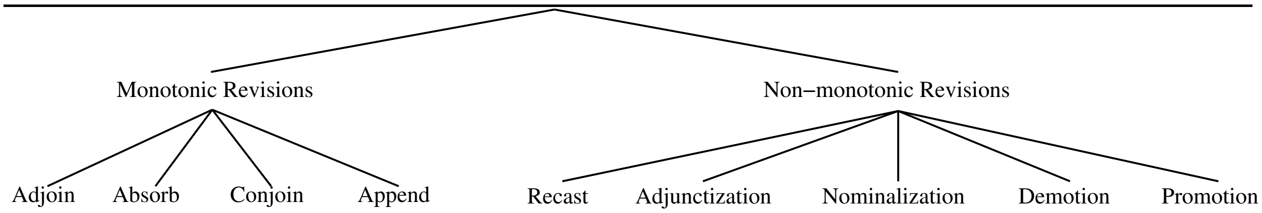

Figure A.1: Revision operation hierarchy: top-levels

- An example of draft and revised phrases illustrating the transformations involved in the revision.
- For each subclass, a surface decrement pair from the basketball corpus (i.e., two semantically and syntactically minimally differing phrases from which the revision operation has been identified).
- When the tool is portable, a corresponding surface decrement pair from the stock market domain.

The last three items are also provided for each side transformation.

A revision schema is a figure with two generic syntactic trees. The left tree shows the draft substructure common to all applications of the revision tool *before* revision (it is called the base structure) and the right tree shows the same substructure *after* revision (it is called the revised structure). The following pictorial conventions are used in these revision schemas:

- Constituents are circles or ovals.
- Structural relationships are lines.
- The lines corresponding to role relations[1] are labeled.
- The Paratactic relation between two constituents is represented by grouping them under a "pseudo-head" tagged by an "&" sign[2].
- The elements **added** by the revision, either constituents or relations, are boldfaced.

Similarly, font conventions are used in the source and target phrase examples from the corpus:

- The **added constituents** are in bold.
- The *deleted constituents* are in italics in the source phrase and, where relevant, their location is marked by the symbol "$\emptyset$" in the target phrase.
- The DISPLACED CONSTITUENTS are in smallcaps (except for numeric constituents which are underlined).
- The <u>constituents which swapped syntactic category</u> are underlined.
- When surrounding context is necessary, the phrase subject to the revision is bracketed.
- Coordinated elements are bracketed.
- The location of implicit constituents (elided, controlled, etc.) is marked the symbol "$\emptyset$".

A.1 Monotonic revisions

Procedurally, a revision tool is defined by a set of transformations to change the draft structure into the revised structure. A monotonic revision tool consists only of an *introductory* transformation, which attaches to the draft the new constituent realizing the additional content while preserving the draft content realization.

As shown in fig. A.1, I identified four main types of monotonic revisions: `Adjoin, Absorb, Conjoin` and `Append`. They differ from each other in terms of either the type of the base structure on which they can be applied or the type of revised structure they produce. I present each of them in turn in the following subsections.

A.1.1 Adjoin

`Adjoin` was already presented in Section 2.5.2.1. It applies only to *hypotactically* structured bases and consists of the introduction of an additional optional constituent A_c. A_c is *adjoined* to the base constituent under the base head B_h. The revision schema of an `Adjoin` is shown in Fig. A.2.

[1] i.e. hypotactic relations, either argument or adjunct.

[2] In coordinations this pseudo-head is realized by a conjunction, in appositions simply by a comma or a similar punctuation mark.

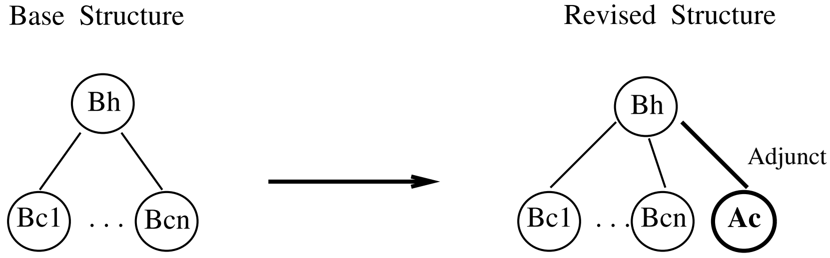

Figure A.2: **Adjoin** schema

The pair of phrases below illustrates how `Adjoin` can be used to add a record information to a game statistic nominal:
`source`: "Armon Gilliam scored [39 points]"
`target`: "Armon Gilliam scored [**a franchise record** 39 points]"

The noun compound "franchise record" (corresponding to A_c in the schema) is simply added as a pre-modifying classifier of the "39 points" nominal (corresponding to B_c in the schema). This is a case of `Adjoin of Classifier`.

`Adjoin` is a versatile tool. The variety of `Adjoin` revision operations is shown in Fig. A.3. The analyzed corpora contained cases where the new constituent was added to a nominal (abbreviated NP in the revision hierarchy) and others where it was added to a clause (abbreviated S in the revision hierarchy). When adjoined to a nominal, the new constituent could fill the following syntactic functions: partitive, classifier, describer and qualifier. For the qualifier syntactic function the added constituent came in two syntactic forms: non-finite clause and relative clause. Finally, a relative clause could express a given type of additional information equally well when adjoined to different draft subconstituents of the same syntactic category (nominal) but embedded at different levels in the draft structure. For example, to add streak information to the draft phrase:
"to power the Golden State Warriors to a 135 119 triumph over the Los Angeles Clippers",

the same revision tool `Adjoin Relative Clause to Nominal` can be applied either to the embedded nominal referring to the losing team (underlined) and yielding:
"to power the Golden State Warriors to a 135 119 triumph over [[the Los Angeles Clippers] , [**who lost for the ninth straight time.**]]"

or alternatively to the top-level nominal conveying the game result (underlined) as a whole and yielding:
"to power the Golden State Warriors to [[a 135 119 triumph over Los Angeles] , [**that extended the Clippers' losing streak to nine games.**]]"

The first revision is thus called `Adjoin of Relative Clause to Bottom-Level Nominal` and the second `Adjoin of Relative Clause to Top-Level Nominal`.

When adjoined to a clause, the new constituent could fill the following syntactic functions: frequency, result, time and co-event, with only a single syntactic category for the adjoined constituent in each case. `Adjoin` was used to add both types of historical information – streaks and records – as well as non-historical information. It was accompanied by three types of side transformations: reference deletion, reference abridging and constituent reordering (see Section A.3 for a discussion of these side transformations).

A surface decrement pair for each subclass of `Adjoin` encountered in the corpora follows:

Adjoin of Classifier

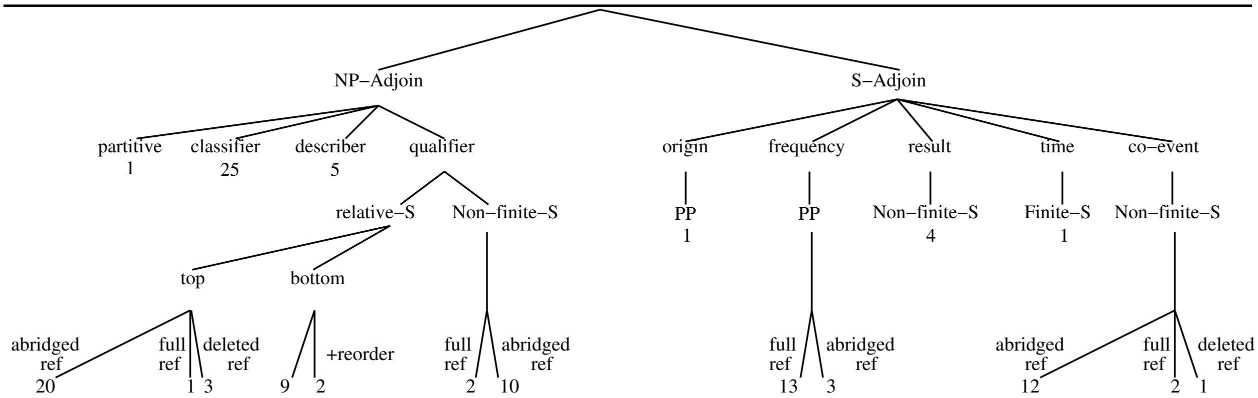

Figure A.3: Adjoin revision operation hierarchy

`source (sports)` : "Patrick Ewing scored [41 points]"

`target (sports)` : "Armon Gilliam scored [**a franchise record** 39 points]"

`source (finance)` : "Volume climbed to [355 million shares]"

`target (finance)` : "Volume climbed to [**a record** 9.091 billion shares]"

Adjoin of Describer

`source (sports)` : "the Boston Celtics"

`target (sports)` : "the **hot-shooting** Boston Celtics"

`source (finance)` : "the Hong Kong market"

`target (finance)` : "the **buoyant** Hong Kong market"

Adjoin of Relative Clause to Top-Level Nominal

`source (sports)` : "to lead the New York Knicks to [a 97 79 victory over the Charlotte Hornets]"

`target (sports)` : "to pace the Houston Rockets to [a 94-83 triumph over the Chicago Bulls **that. completed a two-game sweep**]"

`source (finance)` : "the market got support from [overseas markets]"

`target (finance)` : "the market got an initial boost from [overseas markets, **which proved strong for the second straight day**]"

Adjoin of Relative Clause to Top-Level Nominal with Abridged Reference

`source (sports)` : "to lead the New York Knicks to [a 97 79 victory over *the* Charlotte *Hornets*]" `target (sports)` : "to lead the Cleveland Cavaliers to [a 106 103 win over Detroit **that gave the Pistons their third straight defeat**]"

`source (finance)` : "stock *prices* eased on some profit taking on the Thailand stock exchange"

`target (finance)` : "stocks eased on some profit taking following [the market's recent rally **which lifted prices to five consecutive record close**]"

Adjoin of Relative Clause to Top-Level Nominal with Deleted Reference

`source (sports)` : "to lead the New York Knicks to a 97 79 victory over *the Charlotte Hornets*"

`target (sports)` : "to pace the Atlanta Hawks to [a 113-105 triumph ∅ **that sent the Min-" nesota Timberwolves to their sixth straight defeat**]

Adjoin of Relative Clause to Bottom-Level Nominal

`source (sports)` : "to lead the New York Knicks to a 97 79 victory over [the Charlotte Hornets]"

`target (sports)` : "[the Gold Index] lost 9.8 points to 2,208.2"

`source (finance)` : "to give the Philadelphia 76ers a 95 92 victory over [the Miami Heat **who lost their sixth straight game**]"

`target (finance)` : "[the closely watched German Stock Index, **which gained 6.01 points Monday to set its fifth straight record high**], lost 12.41 points to 1,998.61"

Adjoin of Relative Clause to Bottom-Level Nominal with constituent reordering

`source (sports)` : "and the Jazz routed the Minnesota Timberwolves, <u>110-91</u>"

`target (sports)` : "and the Boston Celtics triumphed <u>119 109</u> over [the Miami Heat **who have lost 11 consecutive road games**]"

Adjoin of Non-finite Clause to Nominal

`source (sports)` : "to lead the New York Knicks to [a 97 79 victory over the Charlotte Hornets]"

`target (sports)` : "to lift the Chicago Bulls to [a 108 93 victory over the Milwaukee Bucks, **running their winning streak to six games**]"

Adjoin of Non-finite Clause to Nominal with Abridged Reference

`source (sports)` : "leading *the* New York *Knicks* to [a 97 79 victory over the Charlotte Hornets]"

`target (sports)` : "lifting New York [a 105 95 triumph over the Los Angeles Lakers, **snapping the Knicks' two game losing skid**]"

Adjoin of partitive to Nominal

`source (sports)` : "Bill Laimbeer scored [25 points]"
`target (sports)` : "Derrick Coleman had [6 **of an NBA team record 22** blocked shots]"

Adjoin of Frequency PP to Clause

`source (sports)` : "as the New York Knicks routed the Philadelphia 76ers 106 79"
`target (sports)` : "as the Phoenix coasted past the Washington Bullets 117 91 **for its fifth straight victory**"
`source (finance)` : "while the Gold Index added 7.0 points to 2,171.3"
`target (finance)` : "while the Amex Market Value Index inched up 0.16 to 481.94 **for its sixth straight advance**"

Adjoin of Origin PP to Clause

`source (sports)` : "Bill Laimbeer scored 25 points"
`target (sports)` : "Dana Barros contributed 21 points **off the bench**"

Adjoin of Frequency PP to Clause with Abridged Reference

`source (sports)` : "to lead *the* New York *Knicks* to a 97 79 victory over the Charlotte Hornets"
`target (sports)` : "to lift New Jersey to a 112 110 triumph over the Jazz, **for the Nets first win at Utah since 1986**"
`source (finance)` : "consumer spending edged up 0.2 percent"
`target (finance)` : "consumer spending rose 0.6 percent **in the index's sixth consecutive rise**"

Adjoin of Co-Event Clause to Clause

`source (sports)` : "helping the Indiana Pacers cruise past the Washington Bullets 131 109."
`target (sports)` : "helping the Los Angeles Clippers defeat the Knicks 101 91, **snapping a 12 game losing streak**"
`source (finance)` : "the key Straits Times Industrials Index soared 108.00 points to 2,302.86 points"
`target (finance)` : "the blue chip Hang Seng Index soared 181.02 points to 9,177.95, **snapping its three session losing streak**"

Adjoin of Co-Event Clause to Clause with Abridged Reference

`source (sports)` : "to lead the New York Knicks to a 97 79 victory over *the* Charlotte *Hornets*"

`target (sports)` : "helping the Milwaukee Bucks to a114 98 victory over San Antonio, **snapping the Spurs eight game winning streak**"

Adjoin of Co-Event Clause to Clause with Deleted Reference

`source (sports)` : "and the Boston Celtics triumphed 119 109 *over the Miami Heat*"

`target (sports)` : "and the Golden State Warriors triumphed 110 105 $\emptyset$ **snapping the Boston Celtics 18 game home winning streak**"

Adjoin of Non-finite Result Clause to Clause

`source (sports)` : "to spark the Chicago Bulls past the Seattle Supersonic 106 100"

`target (sports)` : "leading the Los Angeles Clippers past the Indiana Pacers 122 107 **to snap a seven game losing streak**"

`source (finance)` : "stocks continued to suffer from bouts of profit taking"

`target (finance)` : "the over the counter market caved in to bouts of profit taking **to snap its six session winning streak**"

Adjoin of Time Clause to Clause with (double) abridged reference

`source (sports)` : "to lead *the* New York *Knicks* to a 97 79 victory over *the* Charlotte *Hornets*"

`target (sports)` : "to lead Utah to a 119-89 trouncing of Denver **as the Jazz defeated the Nuggets for the 12th straight time at home**"

`source (finance)` : "*Trading* volume was a busy 198 millions shares"

`target (finance)` : "Volume amounted to a solid 349 million shares **as advances out-paced declines 299 to 218**"

A.1.2 Absorb

The general revision schema of `Absorb` is shown in Fig. A.4. `Absorb` consists of replacing the base constituent B_c by a *new hypotactic* structure in which B_c appears as the jth subconstituent. B_c is thus *absorbed* by this new hypotactic structure, which occupies whatever position B_c occupied in the base structure. `Absorb` differs from the others monotonic revision tools in that it is the only one which results in a revised cluster with an hypotactic relation between the base constituent and the added constituent *with the added constituent as head.*

The pair of phrases below illustrates how `Absorb` can be used to add record information to a game statistic nominal:

`source`: "Larry Bird scored 29 points Monday night including SEVEN 3 POINTERS"

`target`: "Larry Bird scored 29 points Monday night including **matching his own club record with** SEVEN 3 POINTERS"

The nominal "seven 3 pointers" (corresponding to B_c in the schema) is absorbed by the added clauses "matching his own club record" and attached to it via the preposition "with" as an instrument adjunct. The

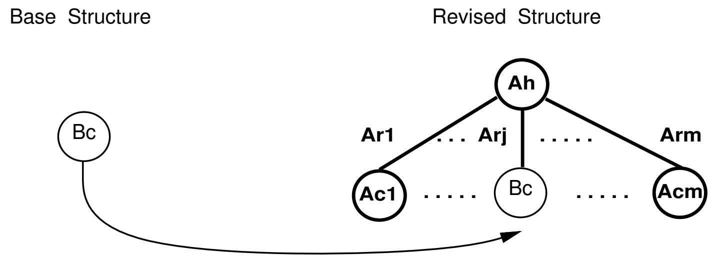

Figure A.4: **Absorb** schema

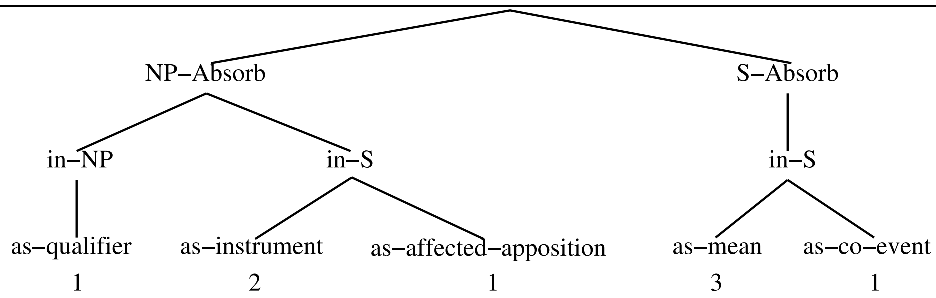

Figure A.5: Absorb revision operation hierarchy

clause resulting from this attachment then gets itself attached under the top-level clause "Larry Bird scored 29 points Monday night including ..." and replaces in this clause the original absorbed nominal "seven 3 pointers". This is thus a case of `Absorb of Nominal into Clause as Instrument Adjunct`.

The variety of `Absorb` revision operations are given in Fig. A.5. Like `Adjoin`, `Absorb` is a versatile tool. The analyzed corpora contained cases where the absorbed constituent was a nominal and others where it was a clause. A nominal could be absorbed either by another nominal or by a clause. When absorbed by another nominal it is always filling the qualifier syntactic function (i.e., the absorbing and absorbed nominals were linked by a preposition). When absorbed in a clause, the nominal could appear either as an instrument adjunct or as an element of an appositive affected argument. In this latter case, absorption is thus indirect via an intermediate paratactic complex: the apposition. In the corpus, all the absorbed clauses were absorbed in another clause, appearing either as a mean adjunct or as a co-event adjunct. Finally, only one absorb subclass was sometimes accompanied by a side transformation (abridged reference): `Absorb of Nominal in Clause as Instrument Adjunct`.

A surface decrement pair for each subclass of `Absorb` encountered in the corpora follows:

Absorb Nominal in Nominal as Qualifier (i.e. as PP complement)

`source (sports)` : "Patrick Ewing scored [41 POINTS]"

`target (sports)` : "Ricky Pierce scored [**a personal season high of** 33 POINTS]"

`source (finance)` : "The blue-chip Financial Times 100-stock index 34.5 points to [3,006.1]"

`target (finance)` : "The blue-chip Financial Times 100-stock index climbed another 38.1 points to [**a new high of** 3,475.1]"

Absorb of Nominal in Clause as element of appositive affected argument

`source (sports)` : "the Los Angeles Clippers recorded a franchise record 68 rebounds in [A 129 112 VICTORY OVER THE DENVER NUGGETS]"

`target (sports)` : "the Houston Rockets held Charles Barkley to 11 in **winning** [[**their fourth straight game**] [A 97 80 TRIUMPH OVER THE PHILADELPHIA 76ERS]]"

Absorb of Nominal in Clause as an Instrument Complement

`source (sports)` : "Johnny Newman scored 30 points Saturday night including [A 12 FOR 12 PERFORMANCE FROM THE FREE THROW LINE]"

`target (sports)` : "Larry Bird scored 29 points Monday night including [**matching his own club record with** SEVEN 3 POINTERS]"

`source (finance)` : "The Korean Composite Index posted [A 6.10 POINT REBOUND]"

`target (finance)` : "Prices plummeted another 26.90 points on Saturday but [**began showing signs of recovery on Monday with** A STEEP 25 POINT REBOUND]"

Absorb of Clause in Clause as Co-Event Adjunct with Agent Control

`source (sports)` : "as THE NEW YORK KNICKS ROUTED THE PHILADELPHIA 76ERS 106 79"

`target (sports)` : "as the Boston Celtics **won their fourth straight game** $\emptyset$ DOWNING THE HOUSTON ROCKETS 108 95"

`source (finance)` : "THE DOW TRANSPORTATION AVERAGE FELL 14.29 POINTS TO 1508.32"

`target (finance)` : "the Dow transportation average **retreated for the second straight session** $\emptyset$ FALLING ANOTHER 13.44 POINTS TO 1495.72"

Absorb of Clause in Clause as Mean Adjunct with Agent Control

`source (sports)` : "helping DETROIT BEAT INDIANA 114 112"

`target (sports)` : "to help the Charlotte Hornets **break a five-game losing streak by** $\emptyset$ HOLDING OFF THE CLEVELAND CAVALIERS 115 107"

`source (finance)` : "THE MARKET ROSE TO 4288.85 POINTS TUESDAY"

`target (finance)` : "the market **tantalized punters by** $\emptyset$ RISING ABOVE 11,000"

A.1.3 Conjoin

`Conjoin` applies to almost any base structure. The only restriction on its application is that the base structure has to belong to a syntactic category that can participate in either a coordination or an apposition. The general revision schema of `Conjoin` is given in Fig. A.6. `Conjoin` consists of replacing the base constituent B_c by a *new paratactic* structure grouping this base constituent with the additional constituent A_c. B_c and A_c are thus *conjoined* in this new paratactic structure. `Conjoin` differs from the two preceding tools (`Adjoin` and `Absorb`) in that it produces a paratactic structure.

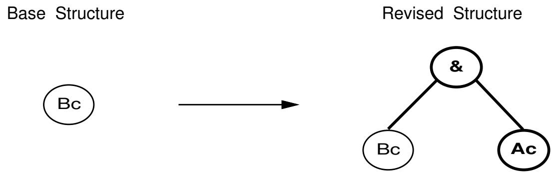

Figure A.6: **Conjoin** schema

The pair of phrases below illustrates how `Conjoin` can be used to add a streak information to a game result clause:
(C_0) "to enable the Philadelphia 76ers [to defeat the Atlanta Hawks 107 103]"
(C_{1_a}) "to enable the Philadelphia 76ers [[to defeat the Atlanta Hawks 107 103] **and** [**win their sixth straight game**]]"

The subordinate clause "to defeat the Atlanta Hawks 107 103" (corresponding to B_c in the schema) becomes the first element of a coordination of two subordinate clauses replacing the original subordinate clause in the main clause. The second clause "win their sixth straight game" (corresponding to A_c in the schema) in the coordination conveys the additional streak information. This is thus a case of `Coordinative Conjoin of Bottom Level Clause`.

The variety of `Conjoin` revision operations is given in Fig. A.7. Both nominals and clauses can be conjoined. The English grammar provides two distinct ways of building a paratactic complex: coordination and apposition. In a coordination, the elements are connected by a coordination conjunction[3] and they can be semantically related in various and potentially loose ways. In an apposition, the elements are simply juxtaposed one after the other and separated only by some punctuation mark. The apposited elements need to be semantic co-referents. The analyzed corpora sublanguage did not contain cases of clausal appositions. As for `Adjoin`, it is possible to add the same information to the draft by conjoining the new constituent with draft constituents located at different depths in the draft structure. For example, consider again adding streak information to the draft phrase C_0 above. Instead of conjoining it with *subordinate* clause "to defeat the Atlanta Hawks 107 103" yielding C_{1_a} above, it is also possible to conjoin it with the *embedding* clause, yielding:
(C_{1_b}) "to [[enable the Philadelphia 76ers to defeat the Atlanta Hawks 107 103] **and** [**record their sixth straight victory**]]"
This is a case of `Coordinative Conjoin of Top Level Clause`. Similarly, the same information can be attached by using apposition on top-level nominals (i.e. those appearing directly as a clause argument or adjunct) or on bottom nominals (i.e. those embedded as nominal modifiers).

Like `Adjoin` and `Absorb`, some subclasses of `Conjoin` are possibly accompanied by reference abridging or deletion as side transformations. Two other side transformations `scope marking` and `verb adjustment` are proper to `Conjoin` (see Section A.3 for a discussion of these side transformations). In the analyzed corpora, `Conjoin` was used to add both historical and non-historical information.

A surface decrement pair for each subclass of Conjoin encountered in the corpora follows:

Top-Nominal Appositive Conjoin

`source (sports)` : "to lead the New York Knicks to [a 97 79 victory over the Charlotte Hornets]"
`target (sports)` : "leading the Denver Nuggets to [[**their first win o the season**] , [a 121 108 victory over the Minnesota Timberwolves]]"
`source (finance)` : "the Amex Market Value Index climbed [4.60 to 477.15]"

[3] i.e. "and", "or" or "but".

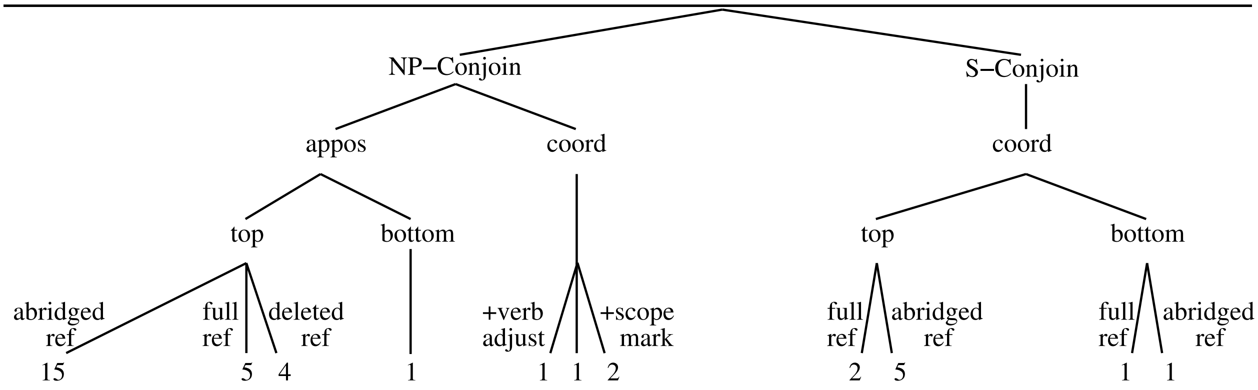

Figure A.7: Conjoin revision operation hierarchy

`target (finance)` : "the Amex Market Value Index inched up [[0.12 to 477.36] – [**its ninth consecutive advance**]]"

Top-Nominal Appositive Conjoin with Abridged Reference

`source (sports)` : "to lead the New York Knicks to [a 97 79 victory over *the* Charlotte *Hornets*]"

`target (sports)` : "lifting the Golden State Warriors to [[a 127 98 victory over Milwaukee] , [**the Bucks' seventh straight road loss**]]"

Top-Nominal Appositive Conjoin with Deleted Reference

`source (sports)` : "to lead the New York Knicks to [a 97 79 victory over *the Charlotte Hornets*]"

`target (sports)` : "to lead the Detroit Pistons to [[**their 11th straight victory over the Sacramento Kings**] , [a 113 110 decision]]"

Bottom-Nominal Appositive Conjoin

`source (sports)` : "to lead the New York Knicks to a 97 79 victory over [the Charlotte Hornets]"

`target (sports)` : "to power the Golden State Warriors to a 135 119 triumph over [[the Los Angeles Clippers] , [**losers of nine straight games**]]"

`source (finance)` : "the market is overcoming the weakness in [IBM]"

`target (finance)` : "weakness in [[IBM] , [**a market leader and key dow component**]]"

Nominal Coordinative Conjoin

`source (sports)` : "a 114 100 victory over Philadelphia , [their first ever win against the 76ers]"

`target (sports)` : "a 118 104 victory over Milwaukee , [[their first ever over the Buck] **and** [**Milwaukee's 10th straight road loss**]]"

`source (finance)` : "durable goods orders slid to a $127.45 billion in July , [the largest decrease since orders fell 5.4% in December 1991]"

`target (finance)` : "durable goods orders rose to $131.59 billion – [[**the index's first gain since February**] **and** [the indicator's largest increase since a 9.1% spurt last December]]"

Nominal Coordinative Conjoin with Scope Marker

`source (sports)` : "leading the Portland Trail Blazers to a 141 125 triumph over the New York Knicks , [their seventh straight victory]"

`target (sports)` : "leading the Utah Jazz to a 108 97 victory over New Jersey , [[their third straight win **overall**] **and** [**10th straight over the Nets**]]"

`source (finance)` : "Alcoa posted [a profit of 57 cents a share]"

`target (finance)` : "the bus company reported [[**a fourth quarter profit of 53% a share**] **and** [**overall** 1992 earnings of $10.9 million or $1.10 a share]]"

Nominal Coordinative Conjoin with Embedding Verb Adjustment

`source (sports)` : "Patrick Ewing *scored* [41 points]"

`target (sports)` : "Armon Gilliam **had** [[24 points] **and** [**11 rebounds**]]"

Top-Clause Coordinative Conjoin

`source (sports)` : "to [lead the New York Knicks to a 97 79 victory over the Charlotte Hornets]"

`target (sports)` : "to [[lead the Cleveland Cavaliers to a 94 78 victory over the Miami Heat] **and** [∅ **break a three game losing streak**]]"

`source (finance)` : "[profit taking sent prices lower on the Thailand Stock Exchange]"

`target (finance)` : "[[a sharp decline in bond prices sent stocks lower] **and** [∅ **halted the Dow's three day record setting march towards the 3800 barrier**]]"

Top-Clause Coordinative Conjoin with Abridged Reference

`source (sports)` : "to [lead the New York Knicks to a 97 79 victory over *the* Charlotte *Hornets*]"

`target (sports)` : "to [[lead the Miami Heat to a 97 79 victory over New Jersey **and snap the Nets three game winning streak**]]"

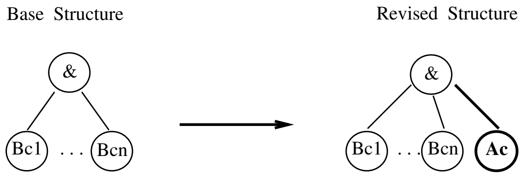

Figure A.8: **Append** schema

Bottom-Clause Coordinative Conjoin

`source (sports)` : "helping Detroit beat Indiana 114 112"
`target (sports)` : "to [[enable the Philadelphia 76ers to defeat the Atlanta Hawks 107 103] **and** [**win their sixth straight game**]]"
`source (finance)` : "as signs of an improving economy forced bound prices to [give up initial gains]"
`target (finance)` : "as computer-guided sell programs and profit-taking teamed up again to [[wipe out initial gains] **and** [**deal the Dow its fourth straight loss**]]"

Bottom-Clause Coordinative Conjoin with Abridged Reference

`source (sports)` : "helping [the Chicago Bulls defeat *the* Seattle *Supersonics* 106 98]"
`target (sports)` : "to help [[the Denver Nuggets defeat Dallas 101 90] **and** [$\emptyset$ **deal the Mavericks their 10th straight loss**]]"

A.1.4 Append

`Append` applies only to draft constituents that are already paratactically structured. As shown in its revision schema Fig. A.8, `Append` simply consists in adding a new element A_c inside such a structure.

The pair of phrases below illustrates how `Append` can be used to add a third game statistic to a nominal conjunction already conveying two statistics by the same player:
"Benoit Benjamin contributed [[19 points] and [16 rebounds]]"
"Benoit Benjamin contributed [[19 points], [16 rebounds] **and** [**six blocked shots**]]"

The nominal "six blocked shots" (corresponding to A_c in the schema) is appended to the nominals "19 points" and "16 rebounds". In the corpora analyzed, `Append` was used with both nominals and clauses. However, it occurred only for coordinations and not for appositions. This is probably due to the general rarity of appositive constructions with more than two elements in English. In some cases, `Append` to Coordinate Clauses was accompanied by an Ellipsis side-transformation (see Section A.3.3 for a discussion of these cases).

The sub-hierarchy of `Append` revision operations is thus trivial and is shown in Fig. A.14 (together with the subhierarchy of `Nominalization` revision operations). In the analyzed corpora, `Append` was used only to add non-historical propositions.

A surface decrement pair for each subclass of Append encountered in the corpora follows:

Append to Coordinate Clauses

`source (sports)` : “[[David Robinson scored 28 points] and [∅ pulled down 19 rebounds]]”
`target (sports)` : “[[Akeem Olajuwon scored 27 points], [∅ grabbed 20 rebounds] **and [blocked four shots**]]”

Append to Coordinate Clauses with Ellipsis

`source (sports)` : “Bill Laimbeer scored 25 points AND Mark Aguirre *added* 24”
`target (sports)` : “Willie Anderson scored 25 points, Terry Cummings ∅ 24 AND **David Robinson 23**”

Append to Coordinate Nominals

`source (sports)` : “Armon Gilliam had [[24 points] and [11 rebounds]]”
`target (sports)` : “Benoit Benjamin contributed [[19 points], [16 rebounds] **and [six blocked shots**]]”

A.2 Non-monotonic revisions

In *non-monotonic* revisions, a single transformation cannot account for the structural differences between the base pattern and the revised pattern. The introductory transformation attaching the new constituent onto the base pattern is accompanied by *restructuring* transformations, necessary to accommodate the new constituent in the revised structure.

As shown in fig. A.1, I identified five main types of non-monotonic revisions: `Recast`, `Adjunctization`, `Nominalization`, `Demotion` and `Promotion`. Each type is characterized by a different set of restructuring transformations which involve displacing base constituents, altering the base argument structure or changing the base lexical head. I present each type of non-monotonic revision in turn in the following subsections.

A.2.1 Recast

A `Recast` is any type of transformation that involves displacing a base subconstituent to accommodate the new content while preserving both the argument structure and the lexical head of the base. The revision schema of `Recast` is shown in Fig A.9. An additional constituent A_c takes the place of a base constituent B_{c_n} filling the role R_n in the draft structure. To accommodate A_c without losing the content conveyed by B_{c_n}, the latter is then moved to fill a new role R_{n+1} in the draft structure.

The pair of phrases below illustrates how `Recast` can be used to add a streak information to a game result clause:
`source`: “Charlotte took A 123 111 VICTORY OVER THE ATLANTA HAWKS”
`target`: “Charlotte took **its third straight win in** A 123 111 VICTORY OVER THE ATLANTA HAWKS”

The additional nominal constituent “its third straight win” realizing the streak information (corresponding to A_c in the schema) is incorporated as the range argument of the verb “to take”. To accommodate this new constituent in this slot, its original occupant, the game result nominal “a 123 11 victory over the Atlanta

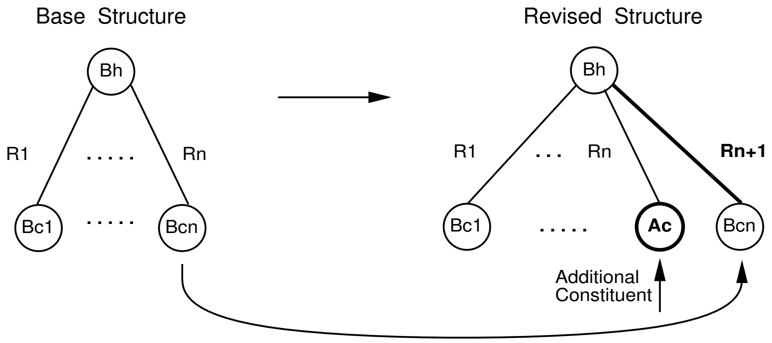

Figure A.9: **Recast** schema

Hawks" (corresponding to B_c in the schema) is displaced as a time adjunct in the target phrase[4]. This is thus a case of `Recast of Range Argument into Location Adjunct`. This example shows that while involving displacement of a base constituent ("a 123 111 victory over the Atlanta Hawks"), Recast preserves both the `process(agent,range)` argument structure of the source phrase and its head verb "to take".

The sub-hierarchy of `Recast` revision operations is shown in Fig. A.10. `Recast` occurs inside both nominals and clauses. Each subclass of `Recast` is characterized by the source role (prefixed by a ">" in Fig. A.10) the recast constituent occupied in the base structure as well as the target role (postfixed by a ">" in Fig. A.10) it occupies in the revised structure. In all nominal cases observed in the analyzed corpora, the source role was the classifier and the target role the qualifier. In clausal cases, there were two source roles, the location and range arguments, and two target roles, the instrument and time adjuncts. In all cases, `Recast` was used to add historical information (either a record or a streak) and was not accompanied by any side transformation.

A surface decrement pair for each subclass of `Recast` encountered in the corpora follows:

Recast in Nominal from Classifier to Qualifier

`source (sports)` : "a 97 79 victory over the Charlotte Hornets"

[4] In this case the time adjunct is introduced by the locative preposition "in" a common case of temporal information conveyed by way of a locative metaphor

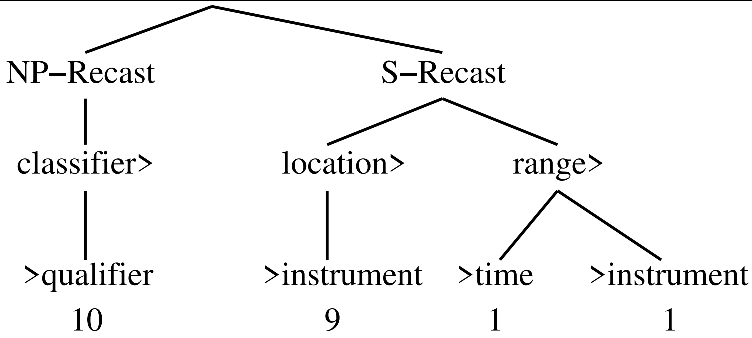

Figure A.10: Recast revision operation hierarchy `target (sports)` : "**their most loop-sided** victory **ever** 122 88 over the Denver Nuggets"

`source (finance)` : "a 12.54 gain"

`target (finance)` : "**its biggest one day** gain **of** 78.64"

Recast in Clause from Location Argument to Instrument Adjunct

`source (sports)` : "to lead the New York Knicks to A 97 79 VICTORY OVER THE CHARLOTTE HORNETS"

`target (sports)` : "leading the Chicago Bulls to **their 23rd straight homecourt victory with** A 131 99 BLOWOUT OF THE MINNESOTA TIMBERWOLVES"

Recast in Clause from Range Argument to Time Adjunct

`source (sports)` : "as Utah took A 124 102 VICTORY OVER THE SAN ANTONIO SPURS"

`target (sports)` : "as Charlotte took **its third straight win in** A 123 111 VICTORY OVER THE ATLANTA HAWKS"

Recast in Clause from Range Argument to Instrument Adjunct with Deleted Reference

`source (sports)` : "to help the Charlotte Hornets post A 104 97 VICTORY *over the Orlando Magic*"

`target (sports)` : "helping the Sacramento Kings post **their first win over the Los Angeles Lakers with** A 102 94 VICTORY"

A.2.2 Adjunctization

`Adjunctization` was already presented in Section 2.5.2.2. It applies only to clausal bases headed by a support verb (i.e, a verb that does not convey in itself any content in the context of the clause, but merely serves as a syntactic support for its meaning-bearing object). The `Adjunctization` schema is given in Fig. A.11. The additional content is realized by a combination of two new constituents: a full-verb V_f (i.e. a verb that bears meaning on its own) and its new object A_c. Deprived of the head verb that supported it in the base clause, original object B_{c_n} migrates to an adjunct position in the revised clause. It has thus been *adjunctized*. `Adjunctization` is similar to `Recast` in that it forces the base constituent to migrate from one slot to another in the clause structure. It differs from `Recast` in that:

- It is does not preserve the base argument structure since a support-verb clause is replaced by a full-verb clause.
- It does not preserve the base lexical head since part of the new content is introduced by a new full-verb.
- It is much more specialized since it applies to nominals nor to clauses headed by full-verbs.

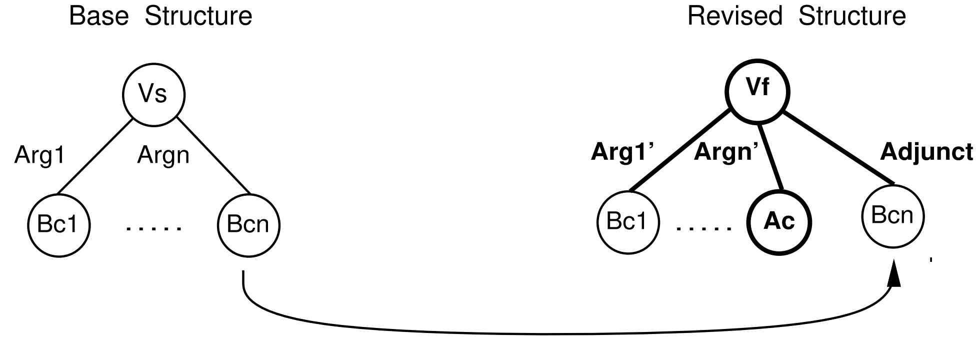

Figure A.11: **Adjunctization** schema

The pair of phrases below illustrates how `Adjunctization` can be used to add a streak information to a game result clause:
`source`: "the Denver Nuggets *claimed* A 124 110 VICTORY OVER THE DALLAS MAVERICKS"
`target`: "the Denver Nuggets **ended their three game losing streak with** A 124 110 VICTORY OVER THE DALLAS MAVERICKS"

The streak information is introduced by a new full-verb "to end" (corresponding to V_f in the schema) and a new object nominal "their three game losing streak" (corresponding to A_c in the schema). Deprived of its support verb "to claim" (corresponding to V_s in the schema), the original object nominal "a 124 1110 victory over the Dallas Mavericks" (corresponding to B_c in the schema) conveying the game result migrates to an instrument adjunct. Since the thematic role of the original object in the source phrase was range, this is a case of `Adjunctization of Range Argument into Instrument`.

The other types of `Adjunctization` are shown in Fig. A.12. Cases of `Adjunctization` are first characterized by the target adjunct role of the displaced constituent. In the analyzed corpora, there were three such target roles: instrument, opposition and destination. Constituents that moved to destination role all originally filled the created role role. Those that moved to opposition role originally filled the affected role. Those that moved to instrument role came from either created, range or location role positions. Some cases of `Adjunctization` were accompanied by side transformations of type reference abridging or reference deletion.

The example below shows an interesting case where `Adjunctization` is coupled with another revision tool, `Demotion` (described on its own in Section A.2.4):
`source`: "to help THE SPURS *post* a 104 97 victory over the Orlando Magic"
`target`: "to help **end** THE SPURS' **five game losing streak with** a 104 86 victory over the Los Angeles Lakers"

In addition to the `Adjunctization` of the game result nominal "a 104 86 victory over the Los Angeles Lakers" from range argument in the source phrase to instrument in the target phrase, the nominal "the Spurs" is also demoted from agent argument in the source phrase to genitive determiner of the affected argument in the target phrase. In this target phrase, the subordinate object clause of "to help" no longer possesses an agent argument of its own like the source phrase. Its agent is an implicit controlled one that is shared with the embedding "help" clause. This example is one of the rare cases of fusion between two classes of revision operations. It could have been equally well classified as a case of `Demotion`.

In the analyzed corpora, `Adjunctization` was used only to add historical information (either records or streaks). It was also the most widely used non-monotonic revision operation.

A surface decrement pair for each subclass of `Adjunctization` encountered in the corpora follows:

Adjunctization of Created Argument into Instrument Adjunct

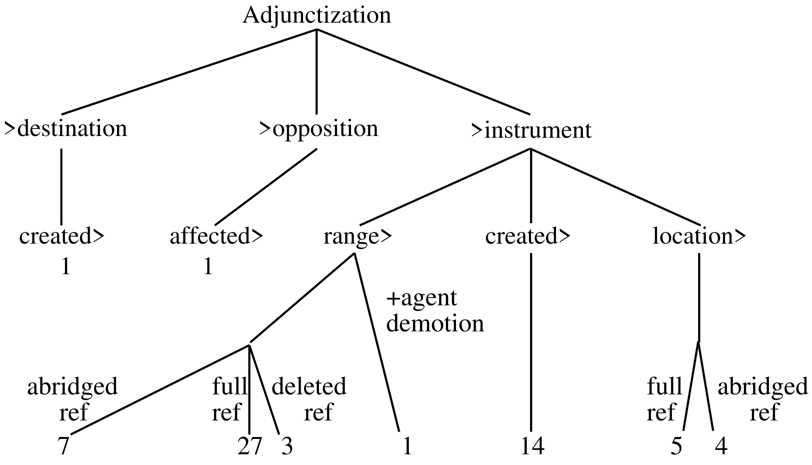

Figure A.12: Adjunctization revision operation hierarchy

`source (sports)` : "Patrick Ewing *scored* 41 POINTS"

`target (sports)` : "Kevin Edwards **tied a career high with** 34 POINTS"

Adjunctization of Range Argument into Instrument Adjunct

`source (sports)` : "to help the Charlotte Hornets *post* A 104 97 VICTORY OVER THE ORLANDO MAGIC"

`target (sports)` : "to help the Milwaukee Bucks **snap a losing streak at five games with** A 95 93 VICTORY OVER THE DETROIT PISTONS"

`source (finance)` : "The Korean Composite Index *posted* A 6.10 POINT REBOUND"

`target (finance)` : "the market **began showing signs of recovery with** A 25 POINT REBOUND"

Adjunctization of Range Argument into Instrument Adjunct with Abridged Reference

`source (sports)` : "to help the Charlotte Hornets *post* A 104 97 VICTORY OVER THE *Orlando* MAGIC"

`target (sports)` : "to help Chicago **remain unbeaten against Minnesota all-time with** A 107 100 VICTORY OVER THE TIMBERWOLVES"

Adjunctization of Range Argument into Instrument Adjunct with Abridged Reference

`source (sports)` : "to help the Charlotte Hornets *post* A 104 97 VICTORY *over the Orlando Magic*" `target (sports)` : "to help the Indiana Pacers **snap the Seattle Supersonics' 10 game winning streak with** A 105 99 TRIUMPH"

Adjunctization of Range Argument into Instrument Adjunct with Demotion of Agent Argument Into Determiner of Affected Argument

`source (sports)` : "to help THE CHARLOTTE HORNETS *post* a 104 97 victory over the Orlando Magic"

`target (sports)` : "to help **end** THE SPURS' **five game losing streak with** a 104 86 victory over the Los Angeles Lakers"

Adjunctization of Location Argument into Instrument Adjunct

`source (sports)` : "and the Chicago Bulls *rolled to* A 128 94 VICTORY OVER THE NEW JERSEY NETS"

`target (sports)` : "and the Denver Nuggets **ended their three game losing streak with** A 124 110 VICTORY OVER THE DALLAS MAVERICKS"

Adjunctization of Location Argument into Instrument Adjunct with Abridged Reference

`source (sports)` : "and the Chicago Bulls *rolled to* A 128 94 VICTORY OVER THE *New Jersey* NETS"

`target (sports)` : "and the San Antonio Spurs **continued their mastery of Dallas with** A 114 107 VICTORY OVER THE MAVERICKS"

Adjunctization of Affected Argument into Opposition Adjunct

`source (sports)` : "Detroit *beat* INDIANA 114 112"

`target (sports)` : "the Chicago Bulls **won their fourth straight** 100 93 OVER THE ATLANTA HAWKS"

Adjunctization of Created Argument into Destination Adjunct

`source (sports)` : "Patrick Ewing *scored* 41 POINTS"

`target (sports)` : "Travis Mays **made 17 of 18 free throws on his way to** 27 POINTS"

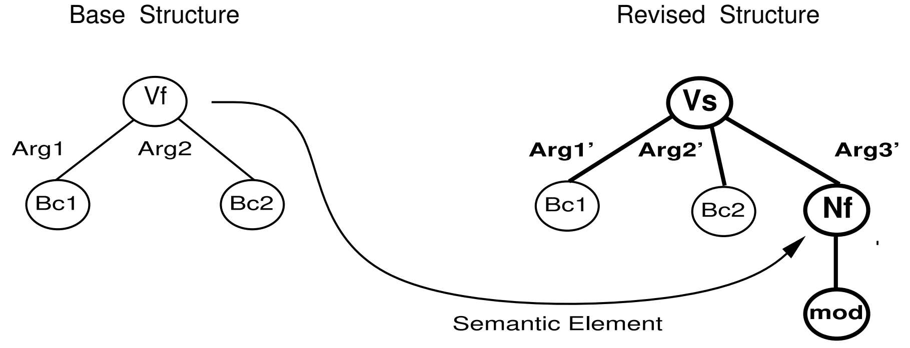

Figure A.13: **Nominalization** schema

A.2.3 Nominalization

`Nominalization` is the converse of `Adjunctization`. `Adjunctization` presented in the previous section applies only to clausal bases headed by a support verb which it replaces by a full verb. Nominalization applies only to clausal bases headed by a full verb V_f which it replaces by a support verb V_s and a new object constituent. This new object is an NP headed by a nominal synonym N_f of the original full verb V_f. The revision schema of `Nominalization` is shown in Fig. A.13. The argument structure of the clause is expanded by the introduction of N_f. This new head noun then serves as the anchor point for the attachment of the constituent A_c conveying the additional content. The nominalization in N_f of the content originally conveyed by V_f allows incorporating the new content as a nominal (instead of clausal) modifier. The main motivation underlying this revision is that nominal modifications are in general more concise that clausal ones.

The pair of phrases below illustrates how `Nominalization` can be used to add a streak information to a game result clause:
`source`: “the Seattle Supersonics <u>defeated</u> the Sacramento Kings 106 91”
`target`: “the Seattle Supersonics **handed** the Sacramento Kings **their 33rd straight** <u>defeat</u> 106 91”

The full verb “to defeat” (corresponding to V_f in the schema) is replaced by a verb-object collocation “to hand a defeat” (“to hand” and “defeat” respectively correspond to V_s and N_f in the schema). The streak information is then attached as a complex ordinal “33rd straight” pre-modifying the head noun “defeat”. Note that the nominal “the Sacramento Kings” has also migrated form direct to indirect object position. In this example, the nominalization is strict: the target noun “defeat” is the precise nominal form of the source verb “to defeat”. This is not necessarily the case. The target noun can be any noun whose meaning in the context of the revised phrase corresponds to the meaning of the source verb in the context of the base phrase. Hence the verb “to beat” expressing the game result in the pattern: <WINNER beat LOSER> can be nominalized by the noun “loss” in the pattern: <WINNER hand LOSER a loss>.

The revision operation of `Nominalization` present here is thus a more general concept than the lexical transformation of nominalization usually described in the linguistic literature.

The sub-hierarchy of `Nominalization` revision operation is shown (together with that of `Append`) in Fig. A.14. Each `Nominalization` subclass is characterized by the type of nominal modifiers used to express the additional content. In the analyzed corpora, an ordinal modifier was always present. It was either alone, accompanied by a classifier or accompanied by a qualifier. `Nominalizations` were not accompanied by any side transformation.

A surface decrement pair for each subclass of `Nominalization` encountered in the corpora follows:

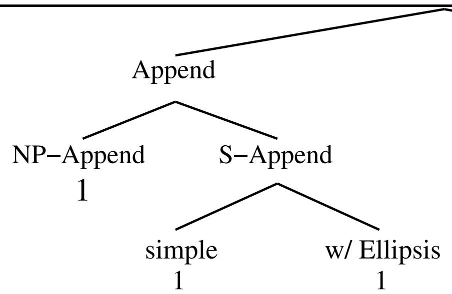

Figure A.14: Append and Nominalization revision operation hierarchy

Nominalization with Ordinal Adjoin

`source (sports)` : "the New York Knicks <u>routed</u> THE PHILADELPHIA 76ERS 106 79"

`target (sports)` : "the Milwaukee Bucks **handed** THE DENVER NUGGETS **their fourth straight** <u>loss</u> 115 98"

`source (finance)` : "the Tokyo stock market <u>plunged</u>"

`target (finance)` : "the Tokyo stock market **posted its third consecutive** <u>decline</u>"

Nominalization with Ordinal and Classifier Adjoin

`source (sports)` : "the New York Knicks <u>routed</u> THE PHILADELPHIA 76ERS 106 79"

`target (sports)` : "the Seattle Supersonics **handed** THE SACRAMENTO KINGS **their 33rd straight road** <u>loss</u> 106 91"

`source (finance)` : the All Industrials Index <u>lost</u> 16.3 points

`target (finance)` : the Hong Kong Stock Exchange **suffered its third straight big** <u>loss</u>

Nominalization with Ordinal and Qualifier Adjoin

`source (sports)` : "the New York Knicks <u>routed</u> THE PHILADELPHIA 76ERS 106 79"

`target (sports)` : "the Phoenix Suns **handed** THE PORTLAND TRAIL BLAZERS **their first** <u>loss</u> **of the season** 121 117"

A.2.4 Demotion

A `Demotion` is any type of transformation that involves displacing a top-level argument of the base structure to an embedded position inside the constituent realizing the additional content in the revised structure. The revision schema of `Demotion` is shown in Fig. A.15. The constituent B_{c_n}, originally a top-level argument in the base structure, gets embedded, in the revised structure, under the head A_{c_h} of the new constituent

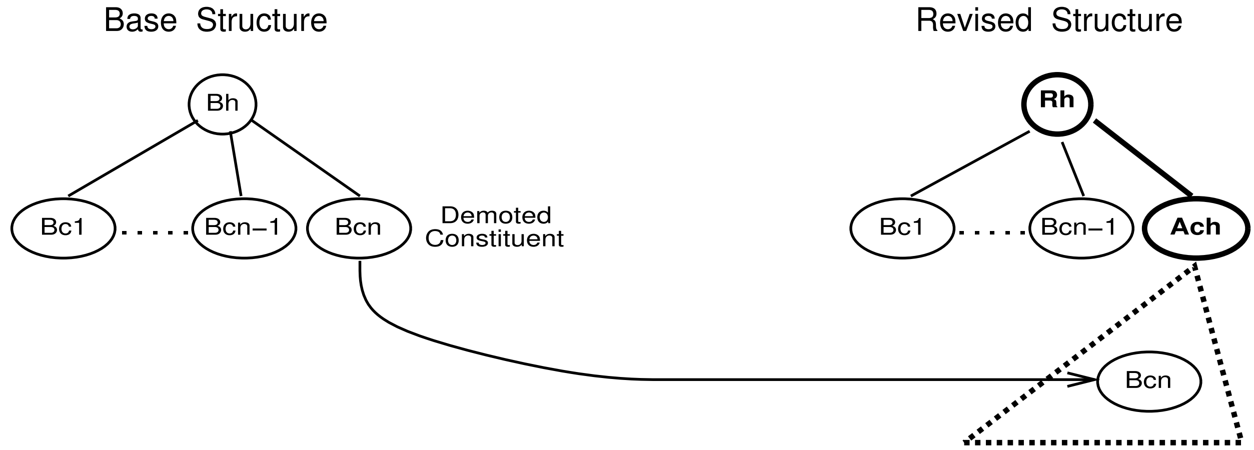

Figure A.15: **Demotion** schema

realizing the additional content. In `Absorb` previously presented, a base constituent also becomes embedded inside the constituent conveying the new content. What distinguishes `Demotion` from `Absorb` is that in the case of `Demotion`:

- The new constituent under which the displaced base constituent gets embedded does not necessarily fill the same role in the revised structure as the one the displaced constituent occupied in the base structure.
- The revised head R_h may differ from the head base B_h.

The pair of phrases below illustrates how `Demotion` can be used to add a record update information to a game statistic clause:
`source`: "Kevin Johnson *scored* 44 POINTS"
`target`: "Kevin Johnson **matched his career high of** 44 POINTS"

The record update information is conveyed by a combination of a new head verb "to match" (corresponding to R_h in the schema) and a new object "his career high" (whose head "high" corresponds to A_{c_h} in the schema). The original object nominal "44 points" (corresponding to B_{c_n} in the schema) is demoted as a qualifier inside the new object constituent. Since the original object filled a Created role in the source phrase and the new one fills an Affected role in the target phrase, this is a case of `Demotion of Created Argument into Qualifier of Affected Argument`.

As shown in Fig. A.16, the analyzed corpora also contained cases where an Affected argument was demoted as the Determiner of a new Affected argument, and cases where a Score phrase was demoted from being an adjunct of the main clause to being an adjunct of a new subordinate Co-Event Adjunct clause. Cases of `Demotion` were not accompanied by any side transformations and were used only to add historical information (either a record or a streak).

A surface decrement pair for each subclass of `Demotion` encountered in the corpora follows:

Demotion of Created Argument into Qualifier of Affected Argument

`source (sports)` : "Patrick Ewing *scored* 41 POINTS"

`target (sports)` : "Kevin Johnson **matched his career high of** 44 POINTS"

`source (finance)` : "the All Singapore Index *closed at* 510.85"

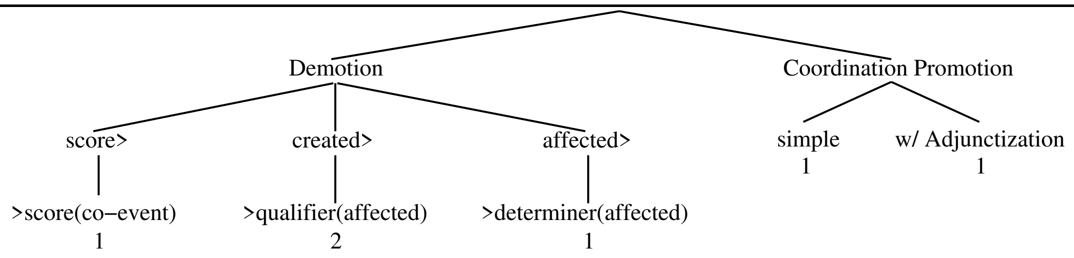

Figure A.16: Demotion and Promotion revision operation hierarchy

`target (finance)` : "the key index **had set a record high of** 7,607"

Demotion of Affected Argument into Determiner of Affected Argument

`source (sports)` : "helping Detroit *beat* INDIANA 131-117"

`target (sports)` : "helping the Phoenix Suns **end** NEW JERSEY **'s six game winning streak** 105 110"

`source (finance)` : "that drove THE MARKET down"

`target (finance)` : "that **snapped** THE DOW **'s three day record setting streak**"

Demotion of Score Phrase from Main Clause to Co-Event Subordinate Clause

`source (sports)` : "the Jazz routed the Minnesota Timberwolves 110 91"

`target (sports)` : "the Orlando Magic shut down the Washington Bullets **for the second time in a week**, **winning** 95 92"

`source (finance)` : "home construction climbed 2.8 percent"

`target (finance)` : "home construction climbed **a second consecutive month during September**, **rising** 2.8 percent"

A.2.5 Coordination Promotion

`Coordination promotion` applies only to hypotactic base structures with a coordinated argument. It undoes the meaning factoring achieved by such structures and transforms it into a top-level coordinated structure of hypotactic elements. The embedded coordination in the base structure is thus *promoted* to the top-level in the revised structure. Accommodating the new content requires undoing the meaning factoring achieved by the embedded coordination when only one of its elements is semantically related to the new content. The revision schema of a `coordination promotion` is shown in Fig. A.17.

The pair of phrases below illustrates how `Coordination Promotion` can be used to add the record equalling nature of a game statistic whose expression in the draft is coordinated with the expression of another game statistic:

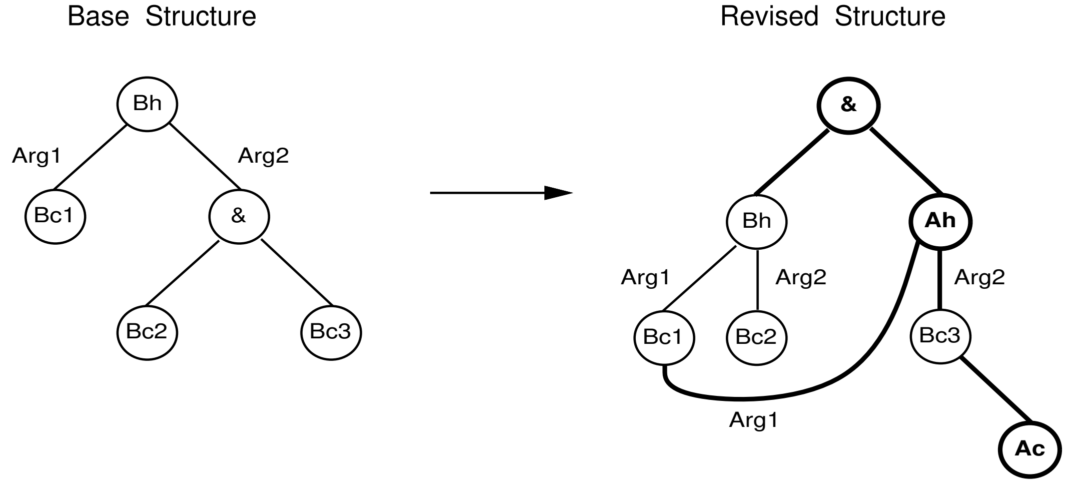

Figure A.17: **Promotion** schema

`source`: "Larry Smith added [[12 points] and [25 rebounds]]"
`target`: "[[Larry Smith added 12 points] and [$\emptyset$ **matched a season high** 25 rebounds]]"

In each phrase the scope of the conjunction is indicated with brackets. In the source phrase, the verb "to add" (corresponding to B_h in the schema) is factored out over the nominal conjunction of "12 points" (corresponding to B_{c_2} in the schema) and "25 rebounds" (corresponding to B_{c_3} in the schema). Without such factoring, the same content would be expressed by a conjunction of clauses, e.g.:
"[[Larry Smith added 12 points] and [$\emptyset$ grabbed 25 rebounds]]".
Because the additional record equalling property concerns only the rebounding statistic and not the scoring one, accommodating this new content requires undoing this factorization and return to the clause coordination form. A second verb "to tie" ("Larry Smith", corresponding to A_h in the schema) can then be added together with a classifier "season high" (corresponding to A_c in the schema) to express this additional content. Since the agent of "to tie" is shared with that of the conjoined verb "to add", it can be elided.

The sub-hierarchy of `Coordination Promotion` revision operation was shown (together with that of `Demotion`) in Fig. A.16. There is only one variant from the simple case illustrated by the example above. In this simple case, after the the introduction of the second verb percolating the coordination up to the top-level, the remaining additional content is incorporated by adjoining a classifier to the object of this second verb. In the complex variant, this additional content is instead incorporated by adjunctizing the object of this second verb as an instrument. On the example source phrase above, this variant yields the following target phrase:
"[[Larry Smith added 12 points] and [$\emptyset$ **matched a season high with** 25 rebounds]]"

`Coordination promotion` is an example of revision operation that is very specialized and involves very complex surface form transformations.

A surface decrement pair for each subclass of Promotion encountered in the corpora follows:

Promotion of Coordination

`source (sports)` : "Kevin Willis contributed [[17 points] and [12 boards]]"
`target (sports)` : "[[Larry Smith added 12 points] **and** [$\emptyset$ **matched a season high** 25 rebounds]]"
`source (finance)` : "the Nikkei traded in range between [[20,686.77] and [20,921.89]]"

`target (finance)` : "[[the Nikkei opened at a low of 20,225.82] and [∅ rose to **its high of** 20,324.63]]"

Promotion of Coordination with Adjunctization of Coordinated Element into Instrument Adjunct

`source (sports)` : "Benoit Benjamin contributed [[19 points], [16 rebounds] and [SIX BLOCKED SHOTS]]"
`target (sports)` : "[[Shaquille O'Neal added [[22 points], [20 rebounds]]] and [∅ **tied a record with** SEVEN BLOCKED SHOTS]]"

A.3 Side transformations

The base pattern transformations presented in the two previous sections meet one of the two following goals:

- Attach the additional constituent(s) realizing the new content (introductory transformation)
- Alter the surface form of the existing content to accommodate this new constituent into the structure under revision (restructuring transformation)

These transformations are the *core* transformations involved in information-adding revisions. However, they are sometimes accompanied by *side* transformations whose goal is to correct whatever verbose repetitions, ambiguous forms or invalid word collocations that the core transformation may have introduced in the draft.

I identified six main types of side transformations: `reference adjustment, argument control, ellipsis, scope marking, ordering adjustment` and `lexical adjustment`. They differ from each other in terms of the goal they satisfy, the aspect of the draft they alter and the types of revision tools that they accompany. I present the six types of side transformations in turn in the following subsections.

A.3.1 Reference adjustment

Reference adjustment, the most widely used side transformation in the analyzed corpora, was already presented in Section 2.5.2.3. Its goal is to insure conciseness by suppressing repetitions introduced by the attachment of the additional proposition. In the analyzed corpora, initial references use a set of default properties associated with the class of the referred entity in the domain ontology. For example, in the sports domain, initial references to teams use both their home and name (e.g., "the New York Knicks"). However, when a revision tool introduces a second reference, this set of properties is then *distributed* between these two references. The most common distribution strategy is to use half the properties in the added reference and then abridge the existing initial reference so that is uses only the remaining half[5]. This is called `Reference Abridging`. The pair of phrases below illustrates how it used as a side transformation to an `Adjoin of Relative Clause to Top Level Nominal` (cf. Section A.1.1 for a discussion of this revision tool):
`source` S_1: "to pace the Atlanta Hawks to a 113 105 triumph over *the* Minnesota *Timberwolves*"
`target` T_1: "to pace the Atlanta Hawks to a 113 105 triumph over ∅ Minnesota ∅, **that sent the Timberwolves to their sixth straight defeat**"

After the `Adjoin` revision introduced a second reference - by name only - to the losing team, the initial

[5]Note that the use of this transformation is based on the assumption that the report targeted audience knows about every property in the default set, since otherwise it could not establish the co-reference link in the revised pattern.

draft reference is abridged from the name + home form to the home only form. In the case of teams in the sports domain, each of the two properties is indifferently used in both the initial draft reference and the reference added by the revision. Thus, a converse target phrase is:
`target` T_2: "to pace the Atlanta Hawks to a 113 105 triumph over the $\emptyset$ Timberwolves, **that sent Minnesota to its sixth straight defeat**"

where the property deleted from the initial draft reference is the home instead of the name.

Without such reference abridging side transformation, the `Adjoin` revision would yield the repetitious forms below (repetitions are underlined):
`target` T_3: ? "to pace the Atlanta Hawks to a 113 105 triumph over the Minnesota Timberwolves, **that sent the Timberwolves to their sixth straight defeat**"
`target` T_4: ? "to pace the Atlanta Hawks to a 113 105 triumph over the Minnesota Timberwolves, **that sent Minnesota to its sixth straight defeat**"
`target` T_5: ? "to pace the Atlanta Hawks to a 113 105 triumph over the Minnesota Timberwolves, **that sent the Minnesota Timberwolves to their sixth straight defeat**"

An alternative strategy to reference abridging is `Reference Deletion`. In this case, the whole set of default referring properties is used in the added reference and to avoid repetition the entire initial draft reference is deleted. On the example source phrase S_1 above, reference deletion yields the following alternative target phrase:
`target` T_6: "to pace the Atlanta Hawks to a 113 105 triumph $\emptyset$ **that sent the Minnesota Timberwolves to their sixth straight defeat**"

Reference deletion is less widely applicable than reference abridging because it requires the semantic link between the draft constituent from which the initial reference is deleted and the constituent added by the revision to be strong enough so that the deleted referent can be inferred from the added context. In the example above, even though the PP referring to the losing team in the game result nominal was deleted, the fact that Minnesota was the loser of the game is still inferable from the added relative clause expressing the losing streak extension

Reference abridging and deletion accompanied several classes of `Adjoin, Conjoin` and `Adjunctization` revisions. Corpora examples for each subclass were given in the previous sections of this appendix.

A.3.2 Argument control

Argument control is a specialized side transformation that systematically accompanies the specific case of `Absorb` where the added absorbing constituent shares one argument with the absorbed draft constituent. In such cases, the common argument in the draft constituent is deleted an is *controlled* by the corresponding argument in the added embedding constituent. This side transformation avoids what would constitute a stylistically inappropriate repetition and opportunistically takes advantage of this commonality between draft and additional content to improve conciseness.

The pair of phrase below illustrates this phenomenon:
`source` S_1: "*Minnesota* ROLLED TO A 106 77 VICTORY OVER THE CHARLOTTE HORNETS"
`target` T_1: "**Minnesota snapped a three game losing streak** $\emptyset$ ROLLING TO A 106 77 VICTORY OVER THE CHARLOTTE HORNETS"

In the target phrase, the top-level reference to Minnesota, which fills the agent argument of the added embedding clause, *controls* the omitted agent argument (marked by the "$\emptyset$" symbol) of the absorbed embedded clause.

Without such a side transformation, the `Absorb` revision would yield the following repetitious target phrase:
`target` T_2: * "**Minnesota snapped a three game losing streak** MINNESOTA ROLLING TO A 106 77

VICTORY OVER THE CHARLOTTE HORNETS"

A.3.3 Ellipsis

Ellipsis is another example of specialized side transformation: it accompanies only Append revisions. It opportunistically improves conciseness by exploiting the fact that the more elements that are present in a coordination, the more fully each subsequent element can be elided. Therefore, when revision is used to append a new element to a coordinated constituent of the draft, part of the coordinated elements can be deleted to become implicit. The pair of phrase below illustrates this phenomenon:
`source` S_1: "Willie Anderson scored 25 points AND Terry Cummings *added* 24 ∅" `target` T_1: "Willie Anderson scored 25 points, Terry Cummings ∅ 24 ∅ AND **David Robinson ∅ 23 ∅**"

In the second element of the source phrase, only the head of its object nominal (realizing the unit of the game statistic) is elided. After the appending of the third element, the verb of the second element becomes implicit as well. In a draft and revision generation framework, there is a significant difference between the verb ellipsis in the second and third element of this target sentence: only the former concerning the existing draft material requires a side transformation; the latter concerns to realization of the added constituent in the revision context. The side transformation Ellipsis presented here is thus a distinct and less general concept than the ellipsis traditionally discussed in the linguistic literature (see the system PLANDOC [McKeown *et al.* 1995] for the generation of this more general type of ellipsis). In particular, it is applicable only to cases where the new constituent shares more than one semantic elements with the draft coordination to which it is appended.

A.3.4 Scope marking

The goal of the three side transformations already presented is to achieve conciseness, by deleting elements in the draft that are redundant with the constituent added by the revision. In some cases where the draft content and the added content share information, the most concise form results from an alternative strategy. This strategy consists of applying the revision tool at the deepest possible level in the draft. However, this may sometimes result in ambiguous phrases. `Scope marking` is a specialized side transformation correcting the ambiguities generated by applying `Coordinative Conjoin` at the deepest possible level.

The example phrases below illustrates the use of Scope Marking with a `Coordinative Conjoin` to add a second game statistic by a different player but of equal value and unit than the first one:
`source` S_1: "Magic Johnson scored 21 points"
`target` T_1: ? "[[Magic Johnson scored 21 points] **and** [**Byron Scott added 21**]]"
`target` T_2: ? "[[Magic Johnson] **and** [**Byron Scott**]] scored 21 points"
`target` T_3: "[[Magic Johnson] **and** [**Byron Scott**]] scored 21 points **apiece**"

T_1 resulting from the application of `Conjoin` at the clause level fails to maximally factors out the shared semantic element between the draft and added content, leading to the repetition of "21 points". T_2 resulting from the application of `Conjoin` at the the clause agent level is more concise. However, it is also ambiguous in that the clause can then be interpreted both distributively (*i.e.,* each scored 21 points) and collectively (*ie..,* they together scored 21 points). In T_3, this ambiguity is removed by the scope marker "apiece" forcing the distributive reading. Note that the side transformation occurs at a higher level in the draft structure than the revision it accompanies. A different scope marker, "total" can be similarly used to force the collective reading. This collective reading can also be forced by adjusting the lexical head of the clause to a verb with intrinsically collective meaning such as "to combine for". This alternative, lexical type of side transformation is presented in Section A.3.6.

A.3.5 Ordering adjustment

As noted in the previous section, a new constituent added to draft by a revision tool can introduce ambiguities. These ambiguities can sometimes be circumvented by simply modifying the precedence relations among the draft constituents without altering their dependence relations. This is especially true for adjuncts whose linear position in the clause is largely underconstrained by syntactic dependencies.

The example phrases below illustrate how reordering constituents can disambiguate a revised phrase resulting from the application of `Adjoin of Relative Clause to Bottom Nominal`:
`source` S_1: “the Boston Celtics triumphed over the Miami Heat 119 109”
`target` T_1: ? “the Boston Celtics triumphed over the Miami Heat **who have lost thee consecutive road games** 119 109”
`target` T_2: “the Boston Celtics triumphed `119 109` over the Miami Heat **who have lost three consecutive road games**”

In T_1, a relative clause is added to the losing team reference to convey a streak information. This additional clause makes the attachment of the score adjunct “119 109” ambiguous: is it the score of only the last game or of each and every game in the streak? In T_2 the first reading is forced by moving the score adjunct in front of the nominal onto which the relative clause was added.

A.3.6 Lexical adjustment

The addition of a new constituent to the draft by a revision tool may introduces ambiguities. It may also violate inter-lexical constraints. Lexical adjustment consists in changing a draft wording not to convey any additional content but rather to avoid either of these two problems.

The pair of phrases below illustrates the use of lexical adjustment with a `Nominal Coordinative Conjoin` to add a rebounding statistic to a scoring statistic by the same player:
`source`: “Armon Gilliam *scored* 24 points”
`target`: “Armon Gilliam **had** 24 points **and 11 rebounds**”

Keeping the verb “to score” after this `Conjoin` would result in the invalid verb-object collocation:
? “to score a rebound”
The embedding head verb is thus adjusted to the versatile verb “to have” which can form a valid verb-object collocation with any noun expressing a game statistic, whether “point”, “rebound”, “assist” etc. Note that, as in the case of scope marking, this lexical adjustment occurs in a higher level draft constituent than the revision tools it accompanies. An example of disambiguating lexical adjustment was given in Section A.3.4. It is interesting to note that four out of six side transformations accompany the two revision tools, `Conjoin` and `Append`, both concerned with coordination. It underlies the easily overlooked complexity of this linguistic phenomenon.

Appendix B

Implementation environment: the FUF/SURGE package

In this appendix, I provide background reference material concerning the implementation of the language generator STREAK (Surface Text Reviser Expressing Additional Knowledge). STREAK summarizes a basketball game by packing all the essential facts about the game as well as their historical significance in a single complex lead sentence. It is based on the draft and revision language generation model put forward in Chapter 3 of this thesis. The present appendix describes the generation tools that underlied the implementation of this model. It presents both these tools as they existed before the development STREAK and the particular extensions to their functionality that this development prompted. The implementation of STREAK itself using these tools was presented in Chapter 4 of this thesis.

STREAK was implemented using the FUF/SURGE package for developing language generation application. This package has been developed by Elhadad over the past 7 years and is still the object of constant improvement and extension. In addition to STREAK, it has been used as the underlying environment for the development of a wide variety of generation applications at Columbia University including multi-media explanation [McKeown *et al.* 1990], stock market reports [Smadja and McKeown 1991], and interactive on-line student advising [Elhadad 1993b]. It has been distributed to over 50 research sites worldwide and is currently being used for the development of 12 projects at seven of these sites, making it the most widely used dedicated package for the development of language generation applications.

FUF [Elhadad 1993b] [Elhadad 1993a] is a special-purpose programming language for language generation based on functional unification. It comes as a package with SURGE, a grammar of English implemented in FUF and usable as a portable front-end for syntactic processing. FUF is the *formalism* part of the package, a language in which to encode the various knowledge sources needed by a generator. SURGE is the *data* part of the package, one already encoded knowledge source usable by any generator. Using the FUF/SURGE package, implementing a generation system thus consists of decomposing *non*-syntactic processing into sub-processes and encoding in FUF the knowledge sources for each of these sub-processes. In the case of STREAK, the decomposition into sub-processes was presented in Chapter 3 of this thesis. In addition to SURGE, STREAK relies on three *non*-syntactic knowledge sources: the phrase planning rule base, the lexicalization rule base and the revision rule base. The encoding of these three knowledge sources in FUF was presented in Chapter 4.

Both FUF and SURGE had to be extended during the development of STREAK to meet some of its special needs. First, the non-monotonicity of STREAK required the implementation of new FUF operators to cut and paste functional descriptions aside from unifying them. These extensions to FUF were implemented by Elhadad. Second, while SURGE already covered a wide variety of syntactic forms for simple clauses[1] it only covered a very few for complex sentences, which aggregate several such clauses. It also did not cover the specialized nominals of quantitative domains. I implemented sizeable extensions to SURGE to attain wide coverage for complex sentences and quantitative nominals. This set of extensions goes far beyond what was

[1] In fact as wide as any other available syntactic processing front-ends for language generation.

needed for the sole needs of STREAK and constitutes in itself a significant, though not central, contribution of this thesis. It was not simply an implementation matter but required to cross-examine the descriptive work of several non-computational linguists and integrate their respective analysis within the unifying computational framework of SURGE. For quantitative nominals, some constructs I observed in newswire report corpora were not mentioned in the linguistic literature and I had to come up with my own analysis. The version of SURGE resulting from these extensions has since been used for other generation applications in addition to STREAK: automated documentation for the activity of telephone network planning engineers [Kukich *et al.* 1994], verbal descriptions of visual scenes [Abella 1994] and generation of business letters.

In what follows, I first give an overview of FUF (Functional Unification Formalism) as a programming language, including the extensions for non-monotonic processing prompted by the development of STREAK. I then describe the initial version of SURGE (Systemic Unification Realization Grammar of English) that forms the core of STREAK's syntactic component. I then present the remaining of this component, *i.e.,* the extensions of SURGE to complex sentences and quantitative nominals, two types of constructs which were pervasive in the corpora of newswire reports that I analyzed for this thesis. Finally, I give the input set used for systematically testing these extensions. This test input set illustrates by example all the new features of the extended coverage of SURGE-2.0.

B.1 The FUF programming language

FUF [Elhadad 1993b] [Elhadad 1993a] is a special-purpose programming language for developing language generation applications. It is an extension (cf. [Elhadad 1990] [Elhadad and Robin 1992]) of the initial functional grammar formalism initiated by [Kay 1979]. Originally, functional grammars where used to implement only two generation subtasks: morpho-syntactic grammaticalization and linearization. This was the case for example in the systems TEXT [McKeown 1985] and TAILOR [Paris 1987]. The extensions to the formalism included in FUF have been in part motivated by its use for a ever wider range of generation subtasks, including lexicalization (in several systems), phrase planning (in ADVISORII) and even non-linguistic tasks like media coordination (in COMET) and ontological deduction (in PLANDOC). The reviser of STREAK constitutes the first use of FUF for implementing a non-monotonic task. The advantage of using FUF to implement deeper aspects of the generation process than just syntax is threefold. First, these deeper aspects can also beneficiate from the strong points of the formalism: a FUF program works with partial information, and is declarative, uniform and very compact (since based on a few powerful concepts such as functional descriptions and unification). Second, using the same formalism to implement all the tasks precludes the need for interfaces reformatting the same information from one formalism used by a given component to a different formalism used by another component. Third, it allows experimenting with different ways to distribute subtask among components at implementation time (for example assigning constituent gaping to the lexicalizer *vs.* to the syntactic grammar). The limitations of FUF were discussed in Section 7.2.1.1.

In what follows, I present only the minimum set of FUF features necessary to provide the reader with the ability to grasp the implementation-related issues of this research as well as to understand the small code excerpts provided as examples in Chapter 4 of this thesis. For an in-depth presentation of FUF, see [Elhadad 1993a] and [Elhadad 1993b].

B.1.1 Functional Descriptions

The basic data structure used in FUF is a Functional Description (FD). An FD describes a set of entities satisfying a set of properties. It consists of a set of pairs `(a v)` called features, where `a` in an attribute and `v` is the value of this attribute for the set of entities described by the FD. `v` can be either an atom or recursively an FD. Allowing recursion makes FDs *structured* representations.

An FD can have any number of features. This variable arity of an FD makes it an *inherently partial* description, or approximation of a particular entity. As more features as added to the FD, the approximation becomes more accurate. This is what makes FDs so attractive for NLP applications: natural language descriptions of a domain entity are also inherently partial. The FD with no feature is noted `nil`. An attribute

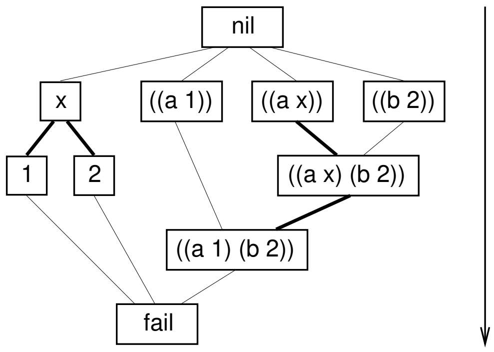

Figure B.1: Space of FDs

with a `nil` value is useless in an FD since it does not provide any information for the attribute *e.g.,*
`((shape round) (size nil)) <==> ((shape round))`.

An attribute can occur only once in a given FD. If it occurred twice with the same value, the second occurrence would bring no additional information and would be useless *e.g.,*
`((shape round) (shape round)) <==> ((shape round))`.
If it occurred twice with different values, the second occurrence would bring contradictory information and be illegal *e.g.,*
`((shape round) (shape square))` is inconsistent.

The basic operation on FDs is unification (noted `u` thereafter). Syntactically it reduces to union for FDs with only atomic values *e.g.,*
`(u ((a 1)) ((b 2)) = ((a 1) (b 2))`
but involves structure mapping with embedded FD values, *e.g.,*

```
(u ((a ((b 1)))        ((a ((b 1)
    (c 2))          =        (e 4))
   ((d 3)               (c 2)
    (a ((e 4)))))       (d 3))
```

Semantically, `(u` $FD_1 FD_2$`)` denotes the intersection of the respective denotations of FD_1 and FD_2. Unification is thus used to gradually provide a description of a particular entity by identifying it as a member of a gradually more restricted set. It defines a partial order among set descriptions. The FD `nil` is the most general description and denotes the universal set of all entities. The FD `fail` is the most specific description. It is equivalent to all inconsistent FDs and denotes the empty set.

This partial order on the space of structured values is paralleled with a partial order on the space of atomic values defined by using the `define-feature-type` construct, *e.g.,* after a

```
(define-feature-type x (1 2))

(u ((a x) (b 2)) ((a 1))) = ((a 1) (b 2))
```

Fig. B.1 gives an example of FD space where the above `define-feature-type` declaration holds. The points shown range from the most general FD, `nil` to the most specific, `fail`. All the direct specialization relations are shown as links between boxes, with those derived from the `define-feature-type` are boldfaced. The other links are derived from unification. The arrow on the left side shows the direction of increasing

```
FD1 structural representation as sublists:
((a ((b ((d 1)
         (e 2)))
     (c 3)))
 (b ((e {a b})))
 (f {b e})
 (h {i}))

FD2 = equation representation equivalent to FD1:
(({a b d} 1)
 ({a b e} 2)
 ({a c} 3)
 ({b e} {a b})
 ({f} {b e})
 ({h} {i}))
```

Figure B.2: FD as equation set

information among these FDs. Types allows the unification process to handle hierarchically organized value sets.

B.1.2 Paths

In addition to an atom or (recursively) a structured FD, the value of a feature in an FD can also be a path to another feature in that FD. A path is a list of attributes surrounded by curly braces. Because each attribute can appear only once at a given depth inside an FD, a path unambiguously identifies a unique feature of the FD. So for example in the FD of Fig. B.2, the path value `{a b d}` unambiguously leads to the value `1`, despite the present of a `b` attribute at the top-level.

A path pointing to a feature whose value is an atom or a structured FD is *direct*. A path pointing to a feature whose value is (recursively) a path is *indirect*. A path pointing to a feature whose value is `nil` (either explicitly or by the absence of the pointed feature from the FD) is *uninstantiated*. So for example, in the FD of Fig. B.2 the path value of the feature under the path `b e` is direct, that of the feature `f` is indirect (and its equivalent direct value is `{a b}` and that of `h` uninstantiated since there is no attribute `{i}` in the FD. In addition to path *values*, FUF also allows path *attributes*. These path attributes allow the representation of an FD as a set of equation. The structural and equative representation of the same FD are compared in Fig. B.2.

Paths encode equality constraints in FUF. This increase of expressibility does not come for free: graphically, paths transforms FDs from trees to general directed graphs[2]. The graphic representation of the FD whose structural and equative representations were given Fig. B.2 is shown in Fig. B.6 of section B.1.3.3.

B.1.3 Functional Grammars

A Functional Grammar (FG) is a meta-FD specifying a set of FDs. An FG represents the set of FDs that it unifies with. The unification of an FD with an FG (noted `uni-fd` thereafter) is a distinct and more complex operation that the unification of two FDs (noted `u`). There is no unification operation for two FGs. `uni-fd` uses `u` as a subroutine. It is more complex than `u` because it handles the meta-attributes `alt`, `cset` and `control`, the meta-value `given` and also *relative* paths, all of which are proper to FGs. These

[2] Potentially cyclic.

```
Initial FD = ((a 1) (b 2))

FG = ((alt (((a ((alt ((2 {c} 1)))))
             (d 3)
             (c 2)))))

Successive values of the FD:
fail                         <------------ next branch in current disjunction
((a {c}) (b 2) (c 1))
((a {c}) (b 2) (c 1) (d 3))
fail                         <------------ backtrack to last choice point
((a 1) (b 2))                <------------ undo intermediate enrichments
((a 1) (b 2) (d 3))
((a 1) (b 2) (c 2) (d 3))
```

Figure B.3: Simple backtracking example

meta-keywords respectively introduce non-determinism, constituent recursion, unidirectional constraints, procedural constraints and local constraints to the unification process. Each subsequent paragraph discusses one of these additions to the expressive power of STREAK.

An FG is a general data structure. A natural language grammar is only one the many types of knowledge that it can conveniently encode. In STREAK, it is used to encode a lexicon, a phrase planning rule base and a revision rule base in addition to a grammar of English. To avoid confusion between the data structure sense and the linguistic sense of the word "grammar", I use "FG" for the former and "syntactic grammar" for the latter.

B.1.3.1 Disjunction and non-determinism

An FD is a conjunction of constraints (each feature defining one constraint). In an FG, it is also possible to define disjunctions of constraints using the meta-attribute `alt` (meaning ALTernation). The value of an `alt` is a list of FDs[3]. Each element in the list corresponds to a *branch* in the disjunction. Disjunctions introduce non-determinism in the unification process.

When an `alt` attribute is encountered in an FG during its unification with an FD, FUF picks one branch in the disjunction, introducing a choice point, and attempts to unify the FD with that branch. If it succeeds, unification proceeds with the next attribute in the FG, temporarily ignoring the remaining branches. However, if a failure later occurs (when an attribute is found both in the FD and the FG with a different value), FUF backtracks to the last choice point and picks another branch in the corresponding disjunction. A simple example of disjunction introducing backtracking in given in Fig B.3. FUF first take the first branch, `(a 2)`, in the disjunction specifying the possible values of `a`. With this choice, unification with the FD fails since `(a 1)` is incompatible with `(a 2)`. FUF then backtracks to its last choice point (in this case, the choice just made) and picks the next branch in the disjunction: `(a c)`. FUF then proceeds enriching the FD with `(c 1)` and then `(d 3)`. When it reaches `(c 2)` however, unification fails again since it is incompatible with the current value `(c 1)`. FUF thus backtracks to its last choice point, the choice of the branch `(a c)` in the `a` disjunction. Before trying the next value, it undoes all the enrichments between the failure and the last choice point. It then picks the third value, proceeds to enrich the FD with `(d 3)` and then `(c 2)` and successfully returns.

Branches of disjunctions introduced by the meta-attribute `alt` are tried in their written order. Another

[3] Only meta-attributes can have *lists* as values. Regular features only atoms and FDs as values.

```
FD = ((x 1)
      (a ((b ((x 2))))))

FG = ((alt (((x 1)                   FG = ((alt (((x 1)
             (c 3)                                 (c 3))
             (cset ({a b})))  <==>                ((x 2)
            ((x 2)                                 (d 4)))
             (d 4)))))               with cat-attribute = x

After top-level unification:
FD = ((x 1)
      (C 3)
      (a ((b ((x 2)))))
      (CSET ({A B})))

After recursive unification:
FD = ((x 1)
      (c 3)
      (a ((b ((x 2)
              (D 4)))))
      (cset ({a b})))
```

Figure B.4: Constituent recursion

meta-attribute, `ralt` (meaning Random ALTernation) allows the definition of disjunctions in which branches are tried in random order. A third meta-attribute `opt` (meaning OPTional) is used as an abbreviation for the simplest types of disjunctions: `((opt fd)) <==> ((alt (FD nil)))`.

FUF's default systematic chronological backtracking strategy can be overwritten by placing control annotations in disjunctions allowing FUF to use a more efficient dependency-directed backtracking strategy (cf. [Elhadad and Robin 1992]). The top-level of an FG is required to be a disjunction.

B.1.3.2 Constituency and recursion

The meta-attribute `cset` (meaning Constituent SET) introduces recursion on constituents. The value of this `cset` attribute is a list of paths. Each path points to a sub-FD called a *constituent*. After the initial unification of the top-level FD has completed, FUF checks whether it has been enriched by a CSET feature. If it has, each sub-FD specified in this feature as a constituent is then recursively unified with the FG. Each sub-FD enriched by this recursion is then replaces the original sub-FD in the top-level FD. An example of recursive unification with one constituent is given in Fig. B.4. At each step, the features added by the last recursion are uppercased.

This recursion mechanism is especially helpful for building a linguistic output, by reaccessing the FG for each linguistic constituent. `cset` allows specifying the constituents in an FD *explicitly*. Alternatively, the constituents can be *implicitly* specified by declaring a given attribute to be the *cat-attribute*[4]. In this case, FUF will recurse on each sub-FD containing this attribute. The default cat-attribute is `cat`. An implicit equivalent of the explicit FG example of Fig. B.4 is given next to it on the same figure.

[4] Thus raising the regular attribute to meta-status.

B.1.3.3 Relative paths and locality

In addition to the *absolute* paths presented in section B.5 which point to a node (in an FD viewed as a graph) by listing the arcs leading to it *from the top-level node*, an FG can also contains *relative* paths which point to a node first indicating how many levels to go up in the graph from the current node (*i.e.,* the node representing the feature whose value is the relative path) and then listing the arcs leading down to the pointed node from that level. A relative path starts with the caret sign (meaning "go up") concatenated with an integer indicating how many levels to go up the structure. It is followed by a list of attributes similar to an absolute path. So for example the relative path `{^2 a b}` means "go up two levels and from there go down following `a` then `b`. The example FD of Fig. B.2 is reproduced here at the top of Fig. B.5 with its corresponding relative path version at the bottom.

The combination of constituent recursion and relative paths provides a powerful abstraction to express local constraints generically. For example in a syntactic grammar number agreement between the determiner and the head of an NP is elegantly encoded in SURGE by the single line: `((determiner ((number {^2 head number}))))`
in the branch of the grammar handling NPs. This prohibits phrase like:
"a victories" or
"several victory"

During the generation of a typical simple clause whose subject and object are both NPs, this branch will be accessed twice, first to perform number agreement of the subject NP and then to perform the number agreement of the object NP. Without relative paths, a distinct line would be needed to encode this agreement for each semantic or syntactic role that an NP can fill in a clause or any other constituent:

```
((determiner ((number synt-roles subject head number))))
((determiner ((number synt-roles object head number))))
((determiner ((number synt-roles indirect-object head number))))
((determiner ((number synt-roles subject qualifier complement head number))))
```

etc.

Since each syntactic category follows many local constraints and can fill a large number of roles, relative paths makes the grammar considerably more compact.

There is a flip side to the expressiveness of relative paths: they can be ambiguous. This is the case when they form a Y configuration with other paths in the graph of an FD. A simple Y configuration is boldfaced in Fig. B.6 showing the graphic form of the FD of Fig. B.5. It makes the relative path in the feature `(f (({^1 b e})))` ambiguous. The attribute `f` leads to the node labeled Y in the figure. At this node there are three possible ways to go up one level before going back down following `b` and then `e`:

- Going up `f` which leads to the top-level and then back to the same Y node following `b` then `e`. This yields the value `((d 1) (e 2))`.
- Taking `b` which leads to the Z node and then following `b` and `e` to the atomic value `2`.
- Taking `e` which leads to the X node and then to nowhere since there is no arc labeled `b` *down* that node. This yields a `nil` value.

When an FG contains such a Y configuration with a relative path, FUF disambiguates it by following up the attribute next to which the relative path appears in the text of the FG. In the example above, this convention means that the first of the three possibilities is chosen.

B.1.3.4 Presence tests and unidirectionality

In general, unification is bidirectional. Depending whether an FG feature is already present in the input FD with which it is unified, the FG can either test part of the *input* or provide part of the *output*. Consider for example the FGs and FDs of Fig. B.7.
The feature `(a 1)` of FG_1 tests part of the input when unified with FD_1, but enriches the output when

```
FD1 contains only absolute paths:
((a ((b ((d 1)
  (e 2)))
     (c 3)))
 (b ((e {a b})))
 (f {b e})
 (h {i}))

FD3 = equivalent to FD1 with relative paths:
((a ((b ((d 1)
  (e 2)))
     (c 3)))
 (b ((e {^2 a b})))
 (f {^1 b e})
 (h {^1 i}))
```

Figure B.5: Relative *vs.* absolute paths

.

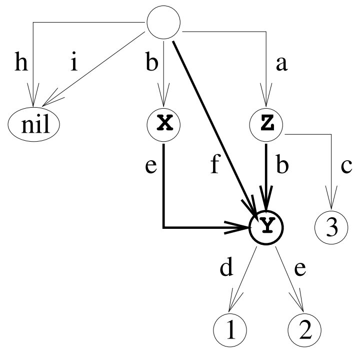

Figure B.6: The FD of figures 1.2 and 1.5 viewed as a graph

```
FG1 = ((alt (((a 1) (b 2)))))
FG2 = ((alt (((a #(under 1)) (b 2)))))

FD1 = ((a 1) (c 3))
FD2 = ((b 2) (c 4))

(u FD1 FG1) = ((a 1) (b 2) (c 3))
(u FD2 FG1) = ((a 1) (b 2) (c 4))

(u FD1 FG1) = ((a 1) (b 2) (c 3))
(u FD2 FG2) = fail
```

Figure B.7: Unidirectional *vs.* bidirectional unification

unified with FD_2. The meta-values `given` and `under` render unification locally unidirectional. For example, in contrast to `(a 1)`, the feature `(a #(under 1))` works only as a test and thus fails with FD_2 which contains no `(a 1)` feature. The meta-value `given` also allows to test for the presence of an attribute in the input but without specifying any value for it. Therefore,
`((a #(under v))) <==> ((a given) (a v))`[5].

B.1.3.5 Procedural tests and negation

An FG can also contain the meta-attribute `control`, which takes as value a call to an arbitrary LISP predicate. The local context of the FG is accessible to this call by way of special relative path parameters prefixed by the macro-character `#@`. These special path parameters are interpreted as pointers inside the sub-FD currently unified with the FG. The LISP predicate is called on the values that these parameters point to inside the FD. So, for example, in Fig. B.8, the `#@{^ a}` parameter points to `3` when unified with FD_1 and to `4` when unified with FD_2 and FD_3.

Like `under`, `control` is an unidirectional attribute, serving only as a test and never as enrichment. It allows mixing the declarative programming paradigm of FUF with the procedural programming paradigm of LISP. It is especially useful to encode inequalities in FUF as shown in Fig. B.8. The FG in that figure enriches the input FD with the feature `(c 1)` if the respective values of `a` and `b` in that FD are *different*. Otherwise, it enriches the FD with the feature `(c 2)`.

B.1.3.6 Modular FGs

The constructs `def-grammar, def-alt` and `def-conj` allow the modular definition and reuse of FGs, disjunctions of FDs and conjunction of features (*i.e.,* FDs). Once defined, the disjunctions are referred to by the `:!` meta-attribute and conjunctions by the `:&` meta-attribute.

Figure B.9 shows a small FG defined in one chunk on the right, with its equivalent modular definition on the left.

[5] Note that an attribute can appear several times in an FG, provided that all its occurrences but one are meta-attributes or paths.

```
FG = ((alt (((control ((not (eq #@{^ a} #@{^ b}))))
             (c 1))
            ((c 2)))))

FD1 = ((a 3) (b 3))
FD2 = ((a 4) (b {a}))
FD3 = ((a 4) (b 3))

(u FD1 FG1) = ((a 3) (b 3) (c 1))
(u FD2 FG1) = ((a 4) (b {a}) (c 1))
(u FD3 FG1) = ((a 4) (b 3) (c 2))
```

Figure B.8: Using `control` to encode negation

.

```
(setf g                                   (def-grammar g
  '((alt (((a b)                            ((alt (((a b)
           (c ((alt (((e f)                         (c ((:! alt1))))
                      (g h))       <==>            ((:& conj1)))))
                     ((i j)))))))
          ((e f)                          (def-alt alt1 (((:& conj1)) ((i j))))
           (g h))))))                     (def-conj conj1 (e f) (g h))
```

Figure B.9: Modular definition of an FG

B.1.3.7 Monotonicity, insert-fd, relocate and canonicity

A central property of the functional unification mechanism onto which FUF is based is that it is *monotonic*. Unification with an FG only *enriches* the input FD with additional features. It does not retract anything: all the features of the input FD are also present in the output FD.

As explained in section 2.5.2.2, many revision operations I observed in human-written summaries are in contrast *not* monotonic. In order to concisely accommodate a new fact, the syntactic structure and lexical material expressing the original content of the draft is altered. Due to their non-monotonic nature, these revisions cannot be implemented in STREAK by relying only on the unification operation. Instead, the FD representation of the draft must be altered by cut and paste operations.

Prior to the development of STREAK, the only user-level functions of the FUF package were the two unification operations:

- `fu` to unify two FDs.
- `uni-fd` to unify an FD and an FG and which recurses on constituents.

In order to extend the usability of FUF to applications, such as STREAK, which uses FDs and FGs as underlying representations, but which requires manipulating them non-monotonically, two additional user-level functions were added: `insert-fd` which pastes a smaller FD as a sub-FD inside a larger one under a given path and the converse `relocate` which cuts a smaller FD appearing inside a larger one as the sub-FD under a given path[6].

The task performed by these two functions is far from being trivial due to the potential presence of paths in the smaller FD. Relocating a sub-FD containing *non-local* paths is especially complex. The difficulties in maintaining path semantics while cutting and pasting a sub-FD with these two functions are illustrated on the simple examples of figures B.10 and B.11.

In Fig. B.10, the FD `smaller` is pasted inside the FD `larger` under the path `{x y}`. The FD `smaller` contains two uninstantiated paths, and one instantiated path, whereas `larger` contains only one path, which is uninstantiated. Recall that paths[7] read from the top-level of FD and their semantics is that of an equation between two features. For example the feature `((f ((g {a e}))))` in `smaller` stands for the equation: (a) `{f g} = {a e}`.

Once `smaller` gets embedded inside `larger` under `{x y}`, the path values inside `smaller` must be prefixed by `{x y}` in order for them to still equate the same feature pairs. For example, in the new global context provided by `larger`, equation (a) above becomes: (b) `{x y f g} = {x y a e}`.

Two system-level FUF functions are used by the unification functions to retrieve the sub-FD under a given path in a larger FD:

- `(top-gdp larger path)`, which stands for Go Down Path from TOP, returns the *value* under `path` in `larger`, *e.g.,* `(top-gdp '((a ((b ((c 1))))) {a b}) = ((c 1))`.
- `(top-gdpp larger path)`, which stands for Go Down Path Pair from TOP, returns the *attribute/value pair* under `path` in `larger`, *e.g.,* `(top-gdp '((a ((b ((c 1))))) {a b}) = (b ((c 1)))`.

In the example run of Fig. B.10 the function calls following the call to `insert-fd` shows that the pasting resulting from that call could not have be done by simply using these two functions in coordination with `fu`. This would fail to change the paths inside `smaller` which would then be erroneously interpreted in the new global context. For example, equation (a) above would become the quite distinct: (c) `{x y f g} = {a e}`. This why a distinct `insert-fd` function is needed.

Note on this example how a pasting operation can instantiates uninstantiated path in both FDs: in `merged1` the values of attributes `i`, `b` and `h` now all points, directly or indirectly, to the atomic value `2`.

[6] Although I participated to their design and testing, these functions were implemented by Elhadad.

[7] At least absolute paths as in the examples at hand.

```
> smaller1
((a ((b {c d})   ;; uninstantiated path
     (e 1)))
 (f ((g {a e})   ;; instantiated path
     (h {i}))))  ;; uninstantiated path

> larger1
((x ((y ((c ((d 2)))
         (i {x y a b})))))  ;; uninstantiated path

> (setf merged1 (insert-fd smaller1 larger1 {x y}))
((x ((y ((c ((d 2)))
         (i {x y a b})           ;; gets instantiated by the pasting of smaller1
         (a ((b {x y c d})       ;; {x y} prefix puts path in right new global context
                                 ;;   in which it gets instantiated
             (e 1)))
         (f ((g {x y a e})       ;; {x y} prefix puts path in right new global context
             (h {x y i}))))))))  ;; {x y} prefix puts path in right new global context
                                 ;;   in which it gets instantiated

> (setf (second (top-gdpp larger1 {x y})) (fu (top-gdp larger1 {x y}) smaller1))
((a ((b {c d})
     (e 1)))
 (f ((g {a e})
     (h {i})))
 (c ((d 2)))
 (i {x y a b}))

> larger1
((x ((y ((a ((b {c d})       ;; points to nowhere in new global context
             (e 1)))
         (f ((g {a e})       ;; points to nowhere in new global context
             (h {i})))       ;; points to nowhere in new global context
         (c ((d 2)))
         (i {x y a b})))))) ;; indirectly points to nowhere
```

Figure B.10: Insert-fd: pasting a smaller FD into a larger FD

```
> larger2
((x ((y ((a ((b {x y c d})   ;; direct path local under {x y}
             (e 1)))
         (f ((g {x y a e})   ;; direct path local under {x y}
             (h {u v i})))   ;; indirect path indirectly local under {x y}
         (c ((d 2)))))))
 (u ((v ((i {x y a b}))))))  ;; indirect path non-local under {u v}

> (setf smaller2 (relocate larger2 {x y}))
((a ((b 2)          ;; canonical new phys rep of class C2b
     (e 1)))        ;; canonical original phys rep of class C1b
 (f ((g {a e})      ;; updated to point to canonical phys rep in new local context
     (h {a b})))    ;; ex indirect, non-local path becomes
                    ;;    direct path to canonical phys rep in new local context
 (c ((d {a b}))))   ;; non-canonical ex phys rep becomes
                    ;;    path to canonical one in new local context

> (setf smaller3 (relocate larger2 {u v}))
((i 2))

> (top-gdp larger2 {x y})
((a ((b {x y c d})  ;; points to nowhere in new local context
     (e 1)))
 (f ((g {x y a e})  ;; points to nowhere in new local context
     (h {u v i})))  ;; points to nowhere in new local context
 (c ((d 2))))       ;; non-canonical phys rep

>
```

Figure B.11: Relocate: cutting a smaller FD from a larger FD

Figure B.11 illustrates the cutting of sub-FDs under two distinct paths inside the same FD. This FD, `larger2`, is similar to `larger1` except that the feature `(i {x y b})` appears under `{u v}` instead of `{x y}`. Its equation set representation is the following:

```
(1a) {x y a e} = 1
(2a) {x y c d} = 2
(3a) {x y a b} = {x y c d}
(4a) {x y f g} = {x y a e}
(5a) {u v i} = {x y a b}
(6a) {x y f h} = {u v i}
```

All the paths in `larger2` are instantiated. In a non-monotonic setting where the value of an attribute can be updated, attributes that happen to share the same value and attributes that are equated by a path have different semantics. For example, in a monotonic setting the two FDs below encode equivalent information:

```
FD1 = ((a 1) (b 1))
FD2 = ((a 1) (b {a}))
```

But this is no longer the case in a non-monotonic setting. If, for instance, the value of `a` is updated to `2`, the value of `b` in FD1 remains `1` while in FD2 it becomes `2`. When cutting a smaller FD inside a larger FD it is thus necessary to preserve the instantiation status of path values.

In that respect, for cutting the sub-FD under `{x y}`, the equations (1a-4a) do not pose any difficulty. The path values in these equations are *local* to the cutting scope `{x y}`. Cutting can thus be done by simply removing the `{x y}` prefix from all the paths involved in these equations. All the paths are still instantiated in the result shown below:

```
(1b) {a e} = 1
(2b) {c d} = 2
(3b) {a b} = {c d}
(4b) {f g} = {a e}
```

Now consider instead, cutting the sub-FD under `{u v}`. Equation (5a) poses a difficulty. The paths on each side do not share a common prefix. The value of attribute `i` is an indirect path ultimately pointing (via `{x y a b}`) to the atomic value `2`. The difficulty lies in the fact that this atomic value is *not* located in the cutting scope `{u v}`. The path value of `i` is thus *non-local*. In such a case, this path value cannot be preserved in the sub-FD that is being cut, since this would means changing what was an instantiated path into an uninstantiated one (put it differently it would mean changing the value of `i` from `2` to `nil`). This path value must thus be replaced by the atomic value to which it ultimately points to. Because of this need to fetch values that are out of the cutting scope, cutting a sub-FD cannot be performed locally by considering only the sub-FD. The whole larger FD needs to be inspected.

Coming back to cutting the sub-FD under `{x y}`, equation (6a) illustrates a further difficulty that indirect paths can trigger. As in the case of (5a), the paths on each side of (6a) do not share a common prefix. The path value of `h` thus at first appears to be *non-local* (like that of `i` in the previous example). This path value is `{u v i}`. The value of `i` is itself a path, `{x y a b}`. In turn the value of `b` is a path, `{x y c d}`. This last path leads to the atomic value, `2`. The important point here is that this chain of path indirections leads back to a location *within* the cutting scope `{x y}`. Thus, in the global context of `larger2`, the path value of `h` really expresses an equality relation between `h` and `d`. Since `d` is under the cutting scope `{x y}` this equality relation is really *local* to this scope and must be preserved in the cut FD. Its locality is only obscured by indirections first out of, but then back in, this cutting scope. Thus, the value of `h` in the sub-FD cut from `larger2` under `{x y}` must be the *path* `{c d}` instead of the *atom* `2`. Path values which do not share a common prefix with the cutting path can therefore *not* be uniformly handled by replacing them by the non-path value to which they ultimately point.

The two examples above show that two types of information, locality and instantiation status, which are hidden by the list or equation set representations of an FD, are crucial to the task of cutting a sub-FD inside

```
smaller2': ((f ((g 1)          ;; non-canonical phys rep of class C2b
                (h 2)))
            (a ((e {f g})      ;; direct path to 1
                (b {f h})))    ;; direct path to 2
            (c ((d {a b}))))   ;; indirect path to 2

smaller2'': ((f ((g 1)         ;; non-canonical phys rep of class C1b
                 (h {c d})))   ;; direct path to 2
             (a ((e {a g})     ;; direct path to 1
                 (b {c d})))   ;; direct path to 2
             (c ((d 2))))
```

Figure B.12: Alternative list representations of the same FD

a larger FD while preserving the semantics of all paths. This suggests that the first stage in the cutting procedure should be a change of representation that explicitly reveals this information. Such representation is the *quotient set*. Each element in a quotient set is a pair whose first element is a set of equivalent paths and whose second element is the non-path value that is commonly pointed to by these equivalent paths. For classes of equivalent *uninstantiated* paths, this second element is `nil`. The quotient set representation of `larger2` is given below:

```
C1a = ([{x y f g},{x y a e}],1)
C2a = ([{x y f h},{u v i},{x y a b},{x y c d}],2)
```

The equality between attributes `h` and `d` that is obscured through indirections in the list representation of `smaller` in Fig B.11, is immediately clear in this quotient set: they belong to equivalent paths. On the list representation, cutting the sub-FD under `{x y}` would involve following path chains while monitoring moves in and out the cutting scope. On this quotient set representation it simply involves: (1) discarding from each class the paths without a prefix matching the cutting scope and (2) removing this prefix from the other paths. The result for `larger2` is shown below:

```
C1b = ([{f g},{a e}],1)
C2b = ([{f h},{a b},{c d}],2)
```

The function `relocate` relies on this quotient set representation to cut a sub-FD inside a larger FD. It thus operates in three steps:

1. Convert the larger FD to quotient set form.
2. Compute the quotient set of the sub-FD under the cutting scope.
3. Convert back the sub-FD quotient set into list form.

The third step is non-trivial due to the fact that, in most cases, there are several possible list representations for each quotient set representation. For example the quotient set above representing the sub-FD cut under `{x y}` inside `larger2`, can be alternatively represented by `smaller2'` and `smaller2''` in Fig. B.12 in addition to `smaller2` in Fig. B.11.

Comparing these three equivalent list representations reveals the sources of multiplicity in the mapping from quotient set representations to list representations:

- The use of indirect paths.
- The location in the list structure of the non-path value associated with each class of equivalent paths.

The particular element in a class of equivalent paths that hold the non-path value of the class is called the *physical representative* of the class.

Allowing `relocate` to pick a *unique* list representation for the sub-FD that it cut from the larger FD using the quotient set representation involves:

- Using only direct paths
- Defining a canonical physical representative for a path class

The choice of canonical physical representative can be left to the particular non-monotonic application for which relocate is needed. A user-provided predicate choosing which of two paths is the most canonical can be passed to `relocate` as an optional parameter. The STREAK implementation uses this facility to define canonicity in a way that reflects the realization relations which hold between the three layers of the draft representation the reviser manipulates. For example, this predicate specifies that the features at the `dss` layer are more canonical than those at the `sss` layer which are in turn more canonical than those at the `dgs` layer.

In the absence of such a user-provided specification of which paths in a class should be its physical representative, `relocate` uses a criteria that optimizes the cut sub-FD for subsequent processing. When unifying two FDs, the FUF interpreter typically spends most of its time following paths. Eliminating indirect paths is already a good source of optimization. Another is to choose the shortest path as the physical representative. For paths of equal length, `relocate` uses the alphabetical order as tie-breaker. This is why the physical representative C1b in `smaller2` is `{a e}` and that of C2a is `{a b}`. The example calls to `relocate` in Fig. B.11, illustrates the fact this function simultaneously performs several tasks:

- Fetching the values of the genuinely non-local paths (*e.g.,* `(i 2)` in `smaller3`).
- Preserving the equality between local attributes that are linked through paths meandering out and back in the cutting scope in the larger FD (*e.g.,* `(h {a b})` in `smaller2`).
- Moving the non-path value of the equivalent path class to its canonical physical representant (*e.g.,* `(b 2)` and `(d {a b})` in `smaller2`).
- Changing all the direct or indirect local paths to the physical representative in the larger FD by direct paths to the canonical physical representative(*e.g.,* `(h {a b})` in `smaller2`).

`relocate` outputs a list that is a *canonical* representation of the corresponding quotient set. This canonical representation is the most efficient among the many possible representations which correspond to the same quotient set. For example, consider unifying the following FD `((c ((d 1))))` with `smaller2` and `smaller2'` respectively. Checking for the compatibility between the value of `d` in this FD and the corresponding value in `smaller2` involves visiting only two nodes `c` and `d`. Doing the same with `smaller2'` requires visiting the nodes `c, d, a, b, f` and `d`. An important side effect of this default definition for the physical representative is that `relocate` called with an empty path parameter can be used to pre-process an FD by putting in the canonical form that is the most efficiently processed by the FUF interpreter. Contrast the output of `relocate` with that of `top-gdp` (with the same FD and path parameters and both given in Fig. B.11), the function used by FUF to retrieve the value of a sub-FD in the monotonic context of unification. It does not put the features of the sub-FD in their new local context nor does it performs any optimization on its path values. The relation between canonicity and efficiency its further discussed in section 7.2.1.1.

B.2 The SURGE-1.0 unification grammar of English

In this section, I present the initial version 1.0 of the syntactic grammar SURGE. This was the version available when I started the implementation of STREAK. The extensions I made to SURGE yield version 2.0 which is presented in the next section. SURGE is a portable front-end for the development of generation application distributed as part of the FUF/SURGE package. FUF is the *formalism* part of the package and can be used to implement various components of a generation system (in the case of STREAK all four components). SURGE is the *data* part of the package: one particular *knowledge source* written in FUF, implementing one reusable component.

SURGE-1.0 was implemented by M.Elhadad and represents his own synthesis, within a single working system of the descriptive work of several linguists, most of which come from the systemic linguistic school and all but one of which non-computational. His main sources of inspiration were [Fawcett 1987] and [Lyons 1977] for the semantic aspects of the transitivity system, [Mel'cuk and Pertsov 1987] for its lexical aspects, [Halliday 1985] and [Winograd 1983] for the other clause systems, the nominal systems and the overall organization of the grammar, [Pollard and Sag 1987] for the treatment of long-distance dependencies and, last but not least, [Quirk *et al.* 1985] for the many linguistic phenomena not mentioned in other works, yet encountered in many generation application domains. In its incomparable comprehensiveness, attention to detail and wealth of examples, [Quirk *et al.* 1985] exemplifies the all too rare type of linguistic work that constitutes a genuine treasure-house for the NLP practitioner. In addition, by being largely neutral with respect to formalism and yet semantically and functionally oriented, it is especially well-suited to a generation perspective. Many of the extensions I made to SURGE-1.0 were inspired by Quirk. The goal of this section is twofold:

- To lay ground for the presentation of this set of extensions in the next section by defining the starting point.
- To provide an implementation-level description of the DGS layer of internal utterance representation that was abstractly described in section 3.3.1.3.

I will therefore not discuss the (significant) portion of SURGE-1.0 that was neither the object of an extension, nor used in the particular sublanguage of STREAK's sports reporting domain. See [Elhadad 1993b] for a discussion of these aspects. In what follows, I first review the generation subtasks carried out by SURGE. I then discuss in some detail the treatment of the clause in general and the transitivity system in particular. I conclude by surveying the treatment of nominals and paratactic complexes.

B.2.1 The task of a syntactic grammar

An input to SURGE is a skeletal lexico-syntactic tree that specifies a sentence to generate by recursively indicating the thematic role organization, open-class lexical items, syntactic categories and associated syntactic features of each constituent. Internally, SURGE is divided into two components: the syntactic grammar proper and the linearizer. The grammar proper is unified with the skeletal sentence specification given in the input to produce a corresponding *enriched* sentence specification that is guaranteed to contain all the information needed by the linearizer to produce a grammatical string of properly inflected English words. Both the grammar proper and the skeletal input are FDs (and so is the enriched specification resulting from their unification).

The nature and scope of this enrichment process is now illustrated on a example sentence of maximum simplicity: "They are winning". The skeletal input accepted by SURGE for that sentence is shown below:

```
((cat clause)
 (process ((type material) (effective no) (lex "win") (tense present-progressive)))
 (partic ((agent ((cat personal-pronoun) (number plural))))))
```

The corresponding enriched description passed to the linearizer is shown in Fig. B.13[8]

[8]This enriched specification was manually post-edited to show only the most essential features. The real output contains

```
((cat simple-clause)
 (generic-cat clause)
 (process ((type material)
           (lex "win")
           (effective no)
           (tense present-progressive)
           (agentive yes)
           (voice active)
           (subcat {oblique})
           (cat simple-verb-group)
           (generic-cat verb-group)
           (dative-move no)
           (insistence no)
           (polarity positive)
           (modality none)
           (person {synt-roles subject person})
           (number {synt-roles subject number})
           (event ((cat verb) (lex {process lex}) (ending present-participle)))
           (be-1 ((tense present)
                  (person {process person})
                  (number {process number})
                  (ending {process ending})
                  (lex "be")
                  (cat verb)))
           (pattern (dots be-1 dots event dots))))
 (partic ((agent ((cat personal-pronoun)
                  (number plural)
                  (generic-cat np)
                  (person third)
                  (synt-function subject)
                  (case subjective)
                  (syntax ((case subjective)
                           (animate yes)
                           (number plural)
                           (person third)
                           (definite yes)))
                  (semantics ((index ((animate yes) (number plural) (person third)))))
                  (reference ((type specific) (possessive no) (interrogative no) (quantitative no)))
                  (pattern (determiner head dots))
                  (head ((cat pronoun)
                         (number plural)
                         (animate yes)
                         (pronoun-type personal)
                         (person third)
                         (case subjective)
                         (lex "they")))
                  (determiner ((cat article-det)
                               (generic-cat det)
                               (head-cat pronoun)
                               (det ((cat article) (lex "")))))
                  (partitive no)))))
 (oblique ((1 {partic agent})))
 (synt-roles ((subject {oblique 1})))
 (verb {process})
 (focus {subject})
 (mood declarative)
 (pattern (dots start {synt-roles subject} dots verb dots)))
```

Figure B.13: Enriched specification for the sentence “They are winning”

The overall task of a syntactic grammar is morpho-syntactic grammaticalization and covers the following subtasks:

- map the thematic structure onto the syntactic structure
- perform agreement
- provide default values for the required syntactic features
- choose closed-class words
- compute precedence constraints
- encode grammatical paraphrasing
- prevent over-generation

Using the enriched specification of Fig. B.13, I now illustrate what each of these subtasks involves.

The roles of the *thematic* structures are semantic, such as `agent`, `process`, `affected` etc. Depending on a variety of other factors such as voice, these thematic roles are mapped onto different roles of the syntactic structure such as `subject`, `verb`, `object` etc. For the simple sentence of Fig. B.13 this mapping is shown in lines 52-54: the thematic structure of this sentence contains only two elements: the `process`[9] described by the sentence and its `agent`. They are respectively mapped onto the two syntactic roles `verb` and `subject`. The process to verb mapping is direct while the agent to subject mapping is indirect through an intermediate *oblique* role (for reasons explained in the overview of SURGE's voice system section B.2.2.2). As most of the processing performed by a syntactic grammar directly acts upon the syntactic roles, thematic structure mapping must be the first task carried out by a syntactic grammar. It is largely independent from the other semantic grammaticalization subtasks and could alternatively be implemented as a separate component. This component would interface a lexical chooser and a syntactic grammar accepting descriptions in terms of syntactic roles. But since thematic role mapping is domain independent, it was integrated into SURGE with the desire to provide a front-end accepting as high-level as possible an input representation while remaining reusable. However, because it is implemented in FUF and thus works with partial information through unification, SURGE accept input descriptions at various levels. It can thus also handle input already specified in terms of syntactic roles, or even as a mix of thematic and syntactic roles. A FUF/SURGE user can thus adjust the level at which SURGE takes over in the generation process, to suit the needs of a particular application.

A syntactic grammar must perform agreement among the sentence constituents. In SURGE, this is done by paths conflating features whose value *must* coincide for a sentence to be grammatical. An agreement resulting from a series of constraint propagations is encoded as a chain of indirect paths. For example, the verb-subject agreement in the simple sentence of Fig. B.13 is encoded in lines 16-17 and 20-21. The verb points to the process whose person and number in turn respectively point to the corresponding features in the subject.

A syntactic grammar must provide a default value (in general, the most common one[10]) for every feature required by the linearizer but not specified in the input. For example the number, person and tense are required for a verb, for the linearizer to be able to properly inflect it. In the example of Fig. B.13, number is propagated from the agent number specified in the input, but person is simply added with its default value: third. These defaults allow for concise input containing only default overriding features. For example, the 7 feature input shown above includes the number feature to override its default singular value with plural.

A syntactic grammar must choose closed-class words such as auxiliaries, pronouns, articles etc. In the three word sentence example at hand, only the main verb "to win" was given in the input. The two others, the auxiliary verb "to be" and the subject pronoun "they" were added by SURGE (cf. lines 23 and 46 in Fig. B.13). While some syntactic features are required to allow the linearizer to choose the right inflection of

about 250 features.

[9] Used here in its systemic linguistic sense to refer to any kind of situation involving participants (abbreviated partic) whether an event, an activity, a state or a relation and thus not carrying any restrictive connotation with respect to aspect.

[10] The notion of "most common" remains intuitive in SURGE, since it it is not based on occurrence counts on large corpora.

an open-class word, others are required to allow the syntactic grammar proper to choose the right function words.

A syntactic grammar must accept and compute precedence constraints among constituents. In SURGE, this is done using the FUF meta-attribute, `pattern`, specifically designed for this very purpose. Examples of patterns are given on lines 25, 39 and 57 of Fig. B.13. The value of this feature is a list whose elements can be either:

- The name of an attribute located at the same level as the pattern feature in the FD structure (e.g., `verb` on line 57).
- A path leading to a feature located at another level than the pattern feature in the FD structure (e.g., `{^ synt-roles subject}` on line 57).
- The wild-card `dots` allowing the order specified by the pattern to be partial (there are three of them on line 57).
- Dummy features, not corresponding to any linguistic constituent, but present as landmarks for the expression of partial ordering constraint (e.g, `start` on line 57).

A special unification procedure is used for these patterns, allowing progressive refinement the order of constituents inside the sentence as constraints on them become available. In Fig. B.13:

- Line 57 encodes precedence among the top-level sentence constituents (subject first, then verb).
- Line 25 precedence inside the verb group (auxiliary = be-1 first, followed by main verb = event)
- Line 39 precedence inside the subject (determiner first - empty in that particular case, cf. line 50 -, the head - the pronoun "they" in that particular case).

When unification finishes, the final complete pattern has been computed. It bears a special meaning to the linearizer. It indicates which among all features of the enriched FD correspond to a syntactic constituent to output in the sentence.

Another task of a syntactic grammar is to encode grammatical paraphrasing. This paraphrasing involves regular syntactic transformations such as passive or dative moves. For example, SURGE can generate the following set of sentences from the same input description in terms of thematic roles:
(1a) "Orlando handed a defeat to Toronto"
(2a) "Orlando handed Toronto a defeat" (dative move)
(3a) "A defeat was handed to Toronto by Orlando" (passive)

A syntactic grammar must encode the syntactic constraints on such paraphrasing to avoid over-generation. For example, suppose that in the input description of the example above, the specification for the "a defeat" NP was changed to simply `((cat personal-pronoun))`. Then SURGE would block the possible choice of the dative move to avoid generating the ungrammatical second form below:
(1b) "Orlando handed it to Toronto"
(2b) * "Orlando handed Toronto it" (dative move)
(2d) "It was handed to Toronto by Orlando" (passive)

B.2.2 The clause sub-grammar

The clause grammar is divided in three main systems:

- The transitivity system which handles the mapping from the thematic structure onto a default syntactic structure for main assertive clauses.
- The voice system which handles departures from the default syntactic structure such as passive, dative moves, clefting, dislocation etc.

- The mood system which handles interrogative, imperative, subordinate and embedded forms.

I overview the SURGE implementation of each of these systems in turn in the following paragraphs.

B.2.2.1 The transitivity system

The thematic roles accepted by SURGE in the input description of a clause are divided into two broad classes: nuclear roles called *participants* (and abbreviated `partic`) and more peripheral ones called *circumstantials* (abbreviated `circum`). Intuitively, participants answer the questions "who/what was involved?" about the process described by the clause, whereas circumstantials answer the questions "when/where/why/how did it happened?". Semantically, processes are classified in a hierarchy of process types[11] and each type is associated with its specialized set of participant roles. In contrast, circumstantial roles are versatile and can attach to processes of virtually any type. Syntactically, three tests can be used to decide whether a particular sentence constituent is a participant or a circumstantial: *Is it movable? Is it obligatory? Can it become subject?*. In general:

- Participants cannot be moved around in the clause without affecting the other elements. Circumstantials can.
- Participants cannot be omitted from the clause while preserving its grammaticality. Circumstantials can.
- Non-subject participants can become subject via transformations such as passivation. Circumstantials cannot.

Several criteria are needed because none of them is always strictly applicable[12]. However, taken together, they allow distinguishing the participants from the circumstantials of almost any clause. Consider, for example, the sentence below:
(1) "Orlando defeated Toronto yesterday". Its process is expressed by the verb "to defeat" and it has three thematic roles respectively expressed by "Orlando", "Toronto" and "yesterday". "Yesterday" can be moved (up-front) without interfering with the rest of the clause whereas "Orlando" and "Toronto" cannot[13]:
(2a) "Yesterday, Orlando defeated Toronto".
(2b) * "Defeated Toronto yesterday, Orlando".
(2c) ? "Toronto, Orlando defeated yesterday".

"Yesterday" can be omitted from (1), whereas "Toronto" and "Orlando" cannot:
(3a) "Orlando defeated Toronto".
(3b) * "Defeated Toronto yesterday".
(3c) * "Toronto defeated yesterday".

Finally, the passive transformation allows "Toronto" to become the subject of the verb "to defeat" but there is no such transformation to do the same with "yesterday":
(4a) "Toronto was defeated by Orlando yesterday".
(4b) * "Yesterday was defeated Toronto by Orlando".

Therefore in (1) above, both "Toronto" and "Orlando" are participants, whereas "yesterday" is a circumstantial.

I now present the set of participants and circumstantials defining the vocabulary of thematic structures accepted as input by SURGE-1.0. Participants are associated with specific types of processes classified in a

[11] The particular hierarchy of process types encoded in SURGE is presented in detail later on in this section.
[12] i.e., there are exceptions for each these rules.
[13] Although (2c) is, strictly speaking, "grammatical", it is appropriate only in very restricted textual contexts, in contrast to (2a) which is appropriate in as many textual contexts as (1).

semantic hierarchy. At the top of this hierarchy, SURGE distinguishes between *simple* and *composite* processes. Simple processes corresponds to clauses with 0 to 2 participants and no causative or resultative meaning. Composite processes correspond to causative or resultative clauses with 2 or 3 participants. In English, a clause can have at most 3 participants.

Figure B.14 shows the sub-hierarchy of simple process types in SURGE, illustrating each participant structure located at the leaf of this hierarchy by an example sentence. Simple processes are first divided into two broad classes: *events* (whether punctual or durative) and *relations* (whether stative or dynamic).

Events are then further divided into *material* events involving entities of the physical realm (either literally or metaphorically), *mental* events involving entities of the cognitive and emotional realms and *verbal* events involving entities of the communicative realm. Operating in orthogonal semantic realms, these three types of events have entirely disjoint sets of participants. There are only material events in the semantic sub-domain of STREAK.

A material event can be either *agentive*, or *effective* or both. It is agentive if the clause presents the event as resulting from the action (intentional or not) of some *agent*. It is effective if the clause specifies the effect of that action. This effect can be either *dispositive* if it affects a pre-existing entity, or *creative* if it creates a new one. All non-agentive material events have a single participant (*Affected* or *Created*). However there is a type of material non-effective events involving two participants. The first is the *Agent* and the second is called the *Range*, because it generally specifies some domain for the activity undergone by the agent. For example in "Tony climbed the mountain", "the mountain" is neither affected nor created by Tony activity but only specifies the range of Tony's climbing. In general, given a clause of the form
"*Subject Verb Object*" the test to decide whether *Object* is a *Range* is the paraphrase
"What *Subject* did to *Object* was to *Verb* it".
This paraphrase is valid for *Affected* and *Created* objects but not for *Range* objects as shown by the non-sensicality of:
"What Tony did to the mountain was to climb it".
When used with very general verbs (*e.g.,* "to make", "to take", "to get", "to have") the *Range* participant specifies the action itself in a nominal form. In that case, the verb and the noun heading the NP filling the *Range* participant are verb-object collocations as in "Michael takes a shower".

Relations are divided into *attributive* relations, linking an entity to one of its property, and *equative* relations, linking two entities. The former ascribes an *attribute* participant (the property) to a *carrier* participant (the entity). The latter identifies one entity (the *identified* participant) by relating it to another entity (the *identifier*). In general, equative relations are reversible either lexically or through passive transformations, whereas attributive relations are not. For example, "Bo owns this car" can be paraphrased by:
"This car belongs to Bo", or by "This car is owned by Bo"
whereas "Bo has long hair" cannot be paraphrased neither by:
"Long hair belongs to Bo" nor by
"Long hair is had by Bo"
.

The example above is a case of *possessive* relation. This type of relation is distinguished because a large class of relational meanings involve ownership, either literally or metaphorically. An even larger class of relational meanings involve a location (also either literally or metaphorically) and are distinguished as *locative* relations. The remaining types of relations are called *ascriptive* and they include copulative clauses. Note that this distinction among *ascriptive*, *possessive* and *locative* relations is orthogonal to the distinction among *attributive* and *equative* relations.

Following a localist approach (cf. [Lyons 1977] pp.718-724), SURGE views existential clauses as expressing a locative relation lacking a location participant. For example, "There is a catch" is analyzed as a syntactic shortcut to the semantically equivalent "There is a catch here" or "There is a catch somewhere". SURGE also classifies as locative, the zero participant clauses describing natural phenomena. For example "It rains" is analyzed as a syntactic shortcut to the semantically equivalent "There is rain falling here". Accompaniment and temporal relations are also analyzed as locative metaphors. Relations that are literally locative are called spatial.

event	material	Agentive non-effective	simple	Michael *Agent*	squats	
			with range	Michael *Agent*	takes	a shower *Range*
		agentive effective	creative	Michael *Agent*	makes	falafels *Created*
			dispositive	Michael *Agent*	eats	falafels *Affected*
		effective non-agentive		Falafels *Affected*	cook	
			creative	Windows *Created*	pop	
	mental			Francisco *Processor*	thinks	
				Francisco *Processor*	liked	how he won *Phenomenon*
	verbal			Michael *Sayer*	speaks	
				Michael *Sayer*	talked	to Cathie *Addressee*
				Michael *Sayer*	said	strange stuff *Verbalization*
relation	ascriptive		attributive	Michael *Carrier*	is	very busy *Attribute*
			equative	The hunter *Identified*	is	the hunted *Identifier*
	possessive			Francisco *Possessor* *Identified*	owns	a big boat *Possessed* *Identifier*
			attributive	Francisco *Possessor* *Carrier*	has	long hair *Possessed* *Attribute*
	locative	natural		It $\emptyset$	rains	
		existential		There $\emptyset$	is	a catch *Located*
		accompaniment		Michael *Located*	is	with his wife *Accompaniment*
		temporal		The end *Located*	is	soon Time
		spatial	attributive	Michael *Located* *Carrier*	is	far away *Location* *Attribute*
			equative	The tree *Located* *Identified*	reaches	the roof *Location* *Identifier*

Figure B.14: Simple process hierarchy in SURGE

Figure B.14 shows the sub-hierarchy of composite processes types in SURGE, again with each participant structure located at the leaf of this hierarchy illustrated by an example. A composite processes results from the superposition within a single syntactic structure of two semantic structures: an event participant structure and a relation participant structure. One syntactic constituent merges these two structures by simultaneously realizing two participants, one from the event structure and one from the relation structure. This meaning superposition is possible because of an underlying cause-effect relationship between the event and relation components of the clause. This compositional and unifying view based on [Fawcett 1987], allows the analysis of explicitly causative clauses (*e.g,* "Mary made him a good man") together with the other types of clauses with three participants (*e.g.,* "Mary gave John a book", "Mary put the book on the table".). It neatly eliminates the need for a *beneficiary* participant[14] which simply becomes both *affected* by the agent's action and the *carrier* of the relation resulting from that action.

There are thus three main criteria to classify composite processes:

- The process type of its event component (in columns 1 and 2 of Fig. B.15).
- The process type of its relation component (in columns 3 and 4 of Fig. B.15).
- Which participants of the event and relation structure are merged into one syntactic constituent (indicated by alignment below each example sentence in Fig. B.15).

The choice of the merging anchor between the two participant structures elegantly accounts for the semantic difference between sentences like:
"He made the Knicks a good team" where the object is both *Affected* and *Carrier*
vs. sentences like:
"He made the Knicks a good coach" where the subject is both *Agent* and *Carrier*

In the current implementation of SURGE, only material events take part in composite processes. A similar compositional approach involving mental and verbal events is possible as well. It would significantly extend the coverage of di-transitive and complex transitive clauses. However, not all such clauses can be analyzed within this framework. This is not a limitation of this particular approach but rather of any approach based on a general participant set. No matter how comprehensive, compositional and well-defined such a general set may be, it is always possible to come up with a particular verb whose argument structure does not neatly fits into it. The immense semantic variety of verbal arguments led to lexicalist approaches (*e.g.,* [Mel'cuk and Polguere 1987]) where no attempt is made to make any generalization across verbs. Each verb is treated as idiosyncratic, with participants *not* semantically labeled and instead given an arbitrary number. However, from a knowledge engineering perspective, it is always best to capture as many generalizations as possible. To do this SURGE relies on general participant structures of figures B.14 and B.15 for most verbs, and falls back on the lexicalist approach for those verbs whose argument structure does not fit any of these participant structures. Since only one verb ("to help") did not correspond to any of SURGE's general participant sets in the target sublanguage of STREAK, I will not discuss SURGE's *lexical processes* here. See [Elhadad 1993b].

The set of circumstantials accepted in input by SURGE-1.0 is given is Fig. B.16. These roles can appear in the input description of a clause independently of its process type. The *Reason, Purpose, Instrument, Manner* and *Accompaniment* circumstantials are based on [Winograd 1983] while *Behalf* comes from [Halliday 1985]. The locative and temporal circumstantials are somewhat ad-hoc. For most of these circumstantials, SURGE-1.0 can generate only prepositional realizations. In Fig. B.16, NP is indicated in parenthesis next to PP, because these prepositional circumstantials are specified in the input as NPs. It is SURGE-1.0 that maps them onto a PP headed by the corresponding default preposition.

The participant sets of SURGE-1.0 allows the generation of a wide variety of both the simple sentences made of a single clause and of the more complex sentences containing embedded clauses in nominal function. The set of circumstantials above also allows the generation of a few cases of the even more complex sentences containing embedded clauses in adverbial function. The variety of semantic relation and syntactic realization of adverbial clauses in English is far greater than what is covered in Fig. B.16. I extensively come back to the limitations of this set of circumstantials in section B.3.2.

[14] Whose definition is always problematic.

Event Type		Relation Type		Example			
agentive	non-effective	ascriptive	attributive	Johnson *Agent* *Carrier*	became	rich *Attribute*	
			equative	The Bulls *Agent* *Identified*	became	the Champs *Identifier*	
		possessive	attributive	Orlando *Agent* *Carrier*	picked	Shaquille *Attribute*	
		locative		Seikaly *Agent* *Carrier*	went	to Miami *Located*	
	dispositive	ascriptive		Riley *Agent*	made	New York *Affected* *Carrier*	a good team *Attribute*
				Riley *Agent* *Carrier*	made	New York *Affected*	a good coach *Attribute*
			equative	The Nets *Agent*	made	Coleman *Affected* *Identified*	the richest *Identifier*
		possessive	attributive	The Nets *Agent*	gave	Coleman *Affected* *Possessor*	more money *Possessed*
		locative		Price *Agent*	threw	the ball *Affected* *Located*	out of bounds *Location*
	creative	ascriptive		Peter *Agent*	brews	his beer *Created* *Carrier*	very strong *Attribute*
		locative		Michael *Agent*	opened	windows *Created* *Located*	on the screen *Location*
non-agentive	dispositive	ascriptive		The game *Affected* *Carrier*	turned	wide open *Attribute*	
		possessive		Coleman *Affected* *Carrier*	received	an offer *Attribute*	
		locative		Coleman *Affected* *Located*	fell	on the floor *Location*	
	creative			The windows *Created* *Located*	popped	on the screen *Location*	

Figure B.15: Composite processes in SURGE

semantic		syntactic	Example	#
class	role	category		
Locative	On-Loc	PP (NP)	Bo kissed her **on the platform**.	a
	In-Loc	PP (NP)	**In Kansas City**, Bo called Gwen.	b
	At-Loc	PP (NP)	**At Kansas City**, Bo called Gwen.	c
	From-Loc	PP (NP)	**From Kansas City**, Bo called Gwen.	d
	To-Loc	PP (NP)	Bo sent the letter **to Kansas City**.	e
Temporal	Time	Adverb	**Yesterday**, Bo triumphed.	f
		PP (NP)	**On Monday**, Bo triumphed.	g
	Background	finite S	**As he received the ball**, Bo smiled.	h
		-ing S	**After receiving the ball**, Bo smiled.	i
		-ed S	**Once injured**, Bo grimaced.	j
Causative	Reason	PP (NP)	**Because of injury**, Bo did not play.	k
		finite S	**Because he was injured**, Bo did not play.	l
	Purpose	PP (NP)	**For enough money**, Bo would play anywhere.	m
		infinitive S	**To gain free-agency**, Bo held out.	n
	Behalf	PP (NP)	**For the Raiders,** Bo scored twice.	o
Determinative	Accompaniment	PP (NP)	**With her**, Bo would go anywhere.	p
Process	Instrument	PP (NP)	Bo pushed him **with both hands**.	q
	Manner	Adverb	**Tenderly**, Bo kissed her.	r

Figure B.16: Circumstantials in SURGE-1.0

B.2.2.2 The voice system

The tables of figures B.14 and B.15 show the diversity of participants associated with the different semantic process types that SURGE accepts as input descriptions. However, for all these process types, each participant ultimately surfaces up as one of the only seven possible syntactic roles allowed by the syntax of English for realizing participants (given here in their partial[15] precedence order):
subject, indirect-object, direct-object, dative-object, passive-object, subject-complement, object-complement.

For each participant set listed in the previous section, SURGE must thus choose a mapping onto a subset of these syntactic roles. This mapping is many-to-many and depends not only on the participants' semantics, but also upon a variety of other factors such as voice, dative moves, clefting and dislocation. SURGE decomposes this complex many-to-many mapping into two stages:

1. A many-to-one mapping from participants to *oblique* roles taking into account only the participants' semantics.
2. A one-to-many mapping from *oblique* to syntactic roles taking into account only the clause structure variations (*i.e.,* voice, dative etc.)

Oblique roles are simply labeled by integers from 1 to 4 reflecting the role's degree of centrality to the clause. For a di-transitive clause in the default active, non-dative form, oblique-1 is subject, oblique-2 is indirect-object and oblique-3 is direct-object. This two-stage mapping is illustrated on the example sentences below:

(1a):	Orlando	defeated	Toronto
	Agent		*Affected*
	Oblique-1		*Oblique-2*
	Subject		*Direct-Object*

[15] The order is partial because, except for the verb, none of these elements is obligatory in every clause.

finite	declarative		*The Knicks won the game*
	interrogative	yes-no	*Did the Knicks win the game?*
		wh	*Who won?*
	bound	nominal	*That the Knicks won the game* was expected
	relative	simple	the game *that the Knicks won*
		embedded	the team *against which the Knicks won the game*
non-finite	imperative		*Win that game!*
	participle	present	The Knicks celebrated *after winning the game*
		past	*Once the game won*, the Knicks celebrated
	infinitive	to	The Knicks are able *to win the game*
		for-to	Ewing must dominate *for the Knicks to win the game*

Figure B.17: Mood in SURGE-1.0

(1b):	Toronto	was defeated	by Orlando
	Affected		*Agent*
	Oblique-2		*Oblique-1*
	Subject		*Passive-Object*

(2a):	Michael	said	something
	Sayer		*Verbalization*
	Oblique-1		*Oblique-2*
	Subject		*Direct-Object*

(2b):	Something	was said	by Michael
	Verbalization		*Sayer*
	Oblique-2		*Oblique-1*
	Subject		*Passive-Object*

The intermediate oblique layer allows to capture the syntactic similarity of the passive transformation from (1a) to (1b) and (2a) to (2b) respectively, despite the semantic difference between the participant sets in these two sentence pairs.

B.2.2.3 The mood system

The mood system deals with the variations in clause form that depends on the type of speech act a clause is to perform, (*i.e.,* whether it is an assertion, a request for action, a request for information etc.) as well as to the position and syntactic function that the clause occupies in the overall sentence structure. As for process types, clausal moods are organized in a hierarchy. Figure B.17 shows the mood hierarchy covered by SURGE-1.0 with an example sentence provided for each leaf element.

The top-level mood distinction is between *finite* clauses, whose verb is conjugated and agrees in person and number with the subject[16], and *non-finite* clauses whose verb is invariant.

Finite clause are further decomposed into *interrogative* clauses mostly used for requesting information, and *declarative*, *bound* and *relative* clauses mostly used for asserting facts. The difference between the latter three lays in their respective position in the sentence structure, namely:

- Main (*i.e.,* the top-level sentence) for declarative mood.
- Subordinated (*i.e.,* modifying another clause) for bound mood.

[16] When the subject is not explicitly present but instead controlled by an embedding clause, the embedded clause verb must agree with the subject of the embedding clause.

- Embedded (*i.e.,* modifying an NP) for relative mood.

Only one type of non-finite clause can be a main clause: *imperative* clauses mostly used for requests. The two other types of non-finite clauses, participle and infinitive clauses are used for assertions and are either subordinated or embedded.

The target sublanguage of STREAK contained only clauses at moods used for assertions. Some of these moods though were not covered by SURGE-1.0. I discuss the extensions I made to SURGE's mood system in section B.3.2.3.

B.2.3 The nominal sub-grammar

The fundamental fact separating nominals from clauses is that the former are considerably more semantically versatile than the latter. There is a nominal paraphrase for virtually any piece of content expressed by any other syntactic category, whereas the availability of clausal paraphrases for content expressed by other categories is the exception. For example nominalizations allow the transformation of the clausal description of an event into a nominal one, *e.g.,*:
"Orlando defeated Toronto" can become
"The defeat of Toronto at the hand of Orlando" or
"Orlando's victory against Toronto" etc.

But there is no clausal form available to simply refer to an entity independently of their taking part in any event or relation as the nominals "Orlando" and "Toronto" do. Beyond nominalization, NPs whose head denote a very broad category like "fact", "event", "property" "place" etc. allow the content range of nominals to include the content range of the other main syntactic category, as illustrated by the examples below:

- "on top of the tree" (PP) $\Longleftrightarrow$ "the place on top of the tree" (NP)
- "Michael is busy" (attributive clause) $\Longleftrightarrow$ "the fact that Michael is busy" (NP)
- "knowledge intensive" (adjectival phrase) $\Longleftrightarrow$ "the property of knowledge intensiveness" (NP)

Because nominals cover such an unrestricted range of meaning, no semantic classification of the entity types they convey has been worked out to the date that matches in comprehensiveness (and could be integrated with) the systemic linguistic semantic classification of the process types that underlies the clause grammar of SURGE. Some interesting investigations in that direction have been carried out by some authors such as [Vendler 1968] or [Levi 1978] but each discuss only a narrow range of nominal meanings and more studies focusing on other ranges are needed before the synthetic integration of these independent efforts within a unifying framework can even be contemplated.

Consequently, at the nominal rank, SURGE accepts descriptions only in terms of syntactic roles (instead of either as thematic or syntactic roles at the clause rank). In a generation application, it thus becomes the responsibility of the lexicalizer to map, in the application's restricted range of nominal meanings, the thematic role substructure it receives as input onto the syntactic role structure to pass to SURGE. How this task is performed in STREAK is discussed in section 4.3.2. The specific syntactic roles accepted by SURGE for nominal description are (given in their partial order of precedence):
Determiner-sequence, Describer, Classifier, Head, Qualifier.

The determiner-sequence is itself decomposed into the following elements:
Pre-determiner, Determiner, Ordinal, Cardinal, Quantifier

The syntactic category accepted as filler for each of these roles by SURGE-1.0, together with an illustrative example nominal is given in Fig. B.18. The constituent filling the syntactic role described at each row is boldfaced in the example.

The order of precedence above is only partial because only the determiner and the head are obligatory. All the others are optional modifiers. They can all co-occur together with one exception: the quantifier is

grammatical function	syntactic category	example
Pre-Determiner	-	**all of** his first ten points
Determiner	article	**a** victory
	demonstrative Pronoun	**this** victory
	question Pronoun	**what** victory?
	possessive Pronoun	**their** victory
	NP	**New York's** victory
Ordinal	simple Numeral	the **third** victory
Cardinal	simple Numeral	**seven** victories
Quantifier	-	Twice as **many** points
Describer	Adjective	an **easy** victory
	present Participle	a **smashing** victory
	past Participle	a **hard fought** victory
Classifier	Noun	a **road** victory
	Adjective	a **presidential** victory
	present Participle	a **winning** streak
Head	common Noun	a **victory**
Qualifier	PP	a victory **over the Nets**
	relative S	a victory **that will be remembered**
	to-infinitive S	a victory **to be remembered**
	for-to-infinitive S	a victory **for them to grab**

Figure B.18: Nominal-Internal grammatical functions and fillers in SURGE-1.0.

exclusive with the ordinal and/or cardinal. The three example nominals below illustrate the simplest and most complex patterns of syntactic roles:

"$\emptyset$	victories"
Determiner	*Head*

"half of	their	first	four	hard fought	overtime	victories	on the road"
Pre-determiner	*Determiner*	*Ordinal*	*Cardinal*	*Describer*	*Classifier*	*Head*	*Qualifier*

"half of	their	many	hard fought	overtime	victories	on the road"
Pre-determiner	*Determiner*	*Quantifier*	*Describer*	*Classifier*	*Head*	*Qualifier*

The same semantic element can be conveyed by different syntactic roles. For example a possessive relation can be expressed by mapping the possessed entity onto the head and the possessor onto either the determiner (genitive case) as in:

the team's	title
Determiner	*Head*
Possessor	*Possessed*

or the qualifier as in:

the title	of the team
Head	*Qualifier*
Possessed	*Possessor*

For the expression of such a possessive relation, this nominal-rank alternation parallels the verb valency alternation at the clause rank, allowing fronting (and thus focusing) different elements of the relation:

The team	owns	the title
Subject	*Verb*	*Direct-Object*
Possessor		*Possessed*

The title	belongs to	the team
Subject	*Verb*	*Direct-Object*
Possessed		*Possessor*

What syntactic role a given subconstituent fills within the overall nominal structure can, in most cases, be straightforwardly determined from its syntactic category and its relative position to the head and/or other modifiers. The non-trivial case arises where a single adjectival pre-modifier occurs between the determiner sequence and the head. Is this adjective a describer or a classifier? The test to take that decision is to attempt to use the pre-modifier in a synonymous attributive clause. If it is also valid in such attributive usage it is a describer, otherwise it is a classifier. So for example, the validity of:
"the victory was easy" tells us that "easy" in "an easy victory" is a describer,
while the invalidity of:
"the victory was presidential" tells us that "presidential" in "a presidential victory" is a classifier.

From a communicative goal perspective, the classifier tends to bring information assumed *known* to the hearer and allowing referent identification, while the describer tends to bring *new* information about a referent already identified by the hearer.

Note that in Fig. B.18 the only possible filler for the head is a *common* noun. This is because SURGE-1.0 treats *proper* nominals as an entirely different syntactic category from the common nominals described so far. These proper nominals can only consist of an unmodified proper noun (*e.g.,* "Orlando"). More complex proper nominals with modifiers (*e.g.,* " streaking Orlando") can therefore not be generated by SURGE-1.0. I come back to this limitation in section B.3.3.

B.2.4 Rest of the syntactic grammar

The only aspect of SURGE-1.0 relevant here outside the clause and nominal sub-grammars is the system dealing with paratactic complexes. This system is orthogonal to the different sub-grammars dealing with the specifics of each syntactic category. It thus treats paratactic complexes generically. They are three types of such complexes in SURGE: conjunctions, appositions and lists.

Conjunctions and appositions satisfy the constraint that each element in the complex needs to be of the same syntactic category[17], whereas the third is for syntactically heterogeneous complexes. Syntactically, conjunctions differ from appositions in that a coordination conjunction links the penultimate element to the last one. In an apposition, only a punctuation mark separate these elements. Semantically, conjunction is a versatile construction that can express a vast array of different relations among its elements including logical, causal, temporal and spatial ones. The precise relation can only be inferred from the context. Apposition requires its elements (rarely more than two) to be co-referent.

Figure B.19 contains an example input description for each types of paratactic complex. It also contains several features of the FUF/SURGE package worthwhile noticing at this point, since they appear again in the STREAK domain examples shown later in this chapter.

The presence of a `complex` attribute indicates that the constituent in question is a paratactic complex and its value specifies which type of complex. The attribute `distinct` contains the list of elements aggregated in the complex. Since in FUF the value of an attribute[18] can only be an atom or recursively a list of attribute-value pairs, lists are internally represented as <car,cdr> pairs. The `''~''` following `distinct` is a FUF macro that renders FDs with list values more legible. For example the FD: `((a ~(1 2)))`
will internally expand as `((a ((car 1) (cdr ((car 2) (cdr none)))))).`

Note how in the apposition example in the center of Fig. B.19 the top-level category `np` is specialized into

[17] Or more precisely of categories related by subsumption since syntactic categories are organized in a hierarchy in SURGE.
[18] Except for *meta*-attributes like `cset`, `pattern`, etc.

```
;; Orlando nipped Toronto and trashed Milwaukee.
((cat clause)
 (complex conjunction)
 (distinct ~(((process ((type material) (lex "nip")))
              (partic ((agent ((cat proper) (lex "Orlando")))
                       (affected ((cat proper) (lex "Toronto"))))))
             ((process ((type material) (lex "trash")))
              (partic ((agent ((cat proper) (lex "Orlando")))
                       (affected ((cat proper) (lex "Milwaukee")))))))))

;; Orlando, the division leader
((cat np)
 (complex apposition)
 (distinct ~(((cat proper) (lex "Orlando"))
             ((cat common) (classifier ((lex "division"))) (head ((lex "leader")))))))

;; a victory over Toronto, that extended Orlando's streak to five games.
((cat np)
 (definite no)
 (head ((lex "victory")))
 (qualifier ((cat list)
             (distinct ~(((cat pp)
                          (prep ((lex "over")))
                          (np ((cat proper) (lex "Toronto"))))
                         ((cat clause)
                          (tense past)
                          (process ((type composite) (relation-type locative) (lex "extend")))
                          (mood relative)
                          (scope {^ partic agent})
                          (partic ((affected ((cat common)
                                              (determiner ((cat proper) (lex "Orlando")))
                                              (head ((lex "streak")))))
                                   (located {^ affected})
                                   (location ((cat pp)
                                              (prep ((lex "to")))
                                              (np ((cat common)
                                                   (cardinal ((value 5)))
                                                   (head ((lex "game"))))))))))))))
```

Figure B.19: Example input descriptions of paratactic complexes in SURGE `proper` and `common` in each element of the apposition. When an element does not contain a `cat` feature it inherits the corresponding feature from the top-level of the complex. This is the case in the clause conjunction example at the top of the figure.

The relative clause of the list example at the bottom of the figure contains the feature `scope`. In a relative clause, one of the semantic participants is the entity that is referred to by the very nominal that the relative clause qualifies. The `scope` feature indicates which participant this is. Without this feature, this input would not unify with SURGE because it lacks the *Agent* participant which is required for composite processes of type material-locative. The fact that the list complex fills a nominal-internal syntactic role is not fortuitous. One of the main usage of `(cat list)` in SURGE is to allow several describers, classifiers or qualifiers that all directly modify the NP head instead of each other.

B.3 SURGE-2.0: extensions to complex sentences and quantitative nominals

In this section, I present the extensions I made to the initial version of SURGE that was presented in the previous section. These extensions were originally motivated by the fact that most sentences observed in the corpus of human-written sports reports contained syntactic constructs beyond the coverage of SURGE-1.0. These constructs fall essentially into two main classes: adverbial relations that link several clauses together inside complex sentences and special nominals that refer to quantities and their historical significance. These gaps in SURGE-1.0's coverage were thus not specific to the particular sports domain used as a testbed for the system implementation aspect of this research. They more generally concerned any application in which either a sizeable variety of complex sentences or comparative references to quantities need to be generated.

In extending SURGE, I did not consider only the set of constructs strictly necessary for the particular application of STREAK. I instead considered the two sets of missing constructs, adverbial complements and quantitative nominals, in a comprehensive manner. For adverbial complements I synthesized several descriptive non-computational linguistic works on these issues within the unifying computational framework of SURGE. For quantitative nominals, faced with the dearth of linguistic work mentioning them, I essentially came up with my own treatment after analyzing their usage in two different domains (sports and the stock market)[19].

The systematic review of these two sets of constructs insured the robustness of the resulting extended version of SURGE. It represents a significant contribution to the growth and maturing of the FUF/SURGE package as a portable syntactic processing front-end for generation, expanding its wide-coverage from the simple clause rank both down to the nominal rank and up to the complex sentence rank. The extended SURGE-2.0 has since been used for the development of several others generation applications at Columbia University including automated documentation [Kukich *et al.* 1994], verbal descriptions of visual scenes [Abella 1994] and business letters. The extensive coverage of adverbial complements provided by SURGE-2.0, proved extremely useful in these three other domains.

In what follows, I start by illustrating the various gaps in the coverage of SURGE-1.0 that were the object of an extension using an example sentence from the corpus of human-written sports summaries. I then review in turn the various extensions done to the clause and nominal sub-grammars of SURGE.

B.3.1 Starting point

Consider the following lead sentence extracted from the corpus of human-written sports summaries[20]:

(0) "*East Rutherford (NJ) – Charles Barkley scored a season high 37 points and Dan Majerle made 6 of 8 three point shots to add 24 Friday night helping the surging Suns pull out a 107-92 victory over the New Jersey Nets losing their third home game in a row against Phoenix.*"

This sentence contains several syntactic constructs not covered by SURGE-1.0. First consider the adverbial infinitive clause "*helping ... in a row*" forming the entire second half of the sentence. It cannot be generated by SURGE-1.0 for two different reasons:

- The semantic relation that this attachment realizes. The only types of subordinate adverbial present participle clauses covered by SURGE-1.0 are those whose function is to indicate the time of the actions conveyed by the matrix. The "*helping ... in a row*" dependent clause does not match this type of clause neither semantically, since this time is indicated by the NP "*Friday night*" just preceding this dependent clause, nor syntactically, since it does not start by a temporal subordinating conjunction (*e.g.*, "*after*", "*before*") as required of temporal background clauses.

[19] Numerous discussions with M.Elhadad and B.Passoneau about these constructions have tremendously helped form the analysis I propose here.

[20] Strictly speaking, this sentence did not actually occur in the corpus. For the sake of presentation, it was rather reconstructed from pieces of different sentences, each of them actually occurring in the corpus. This reconstruction conveniently allows every class of syntactic constructs that required an extension to SURGE-1.0 to be illustrated on this single example

- The nature of the matrix constituent onto which it is attached. It is not solely Majerle's performance that helped the Suns win but rather both Barkley's and Majerle's performances. The "*helping ...*" adverbial clause therefore modifies the conjunction of clauses forming the entire first half of the sentence: "*Charles Barkley ... and Dan Majerle ... adding 24*". In SURGE-1.0 adverbial elements could only separately modify individual clauses, not globally modify paratactic complexes.

Therefore, both the paratactic complex sub-grammar and the part of the transitivity system handling adverbial roles in the clause sub-grammar must be extended. The direct object of this "*helping ... in a row*" dependent clause is itself a clause: "*pull out ... in a row*". This *bare* variety of infinitive clause which lacks the frontal "*to*", was not covered by SURGE-1.0. The mood system of the clause sub-grammar must therefore be extended as well.

Now consider the referring expressions in sentence (0). The losing team is referred to as: "*the New Jersey Nets losing their ...* ", where the proper noun "*New Jersey Nets*" is post-modified by the qualifying present participle clause "*losing their ...*". SURGE-1.0 did not cover nominals of this form since: (1) it allowed only the relative and infinitive mood for qualifying clauses and more importantly (2) it did not allow modification of proper nouns. Note also the *pre*modifier in the first reference to the winning team: "*the surging Suns*". This winning team is mentioned twice, but only half of its complete proper noun is used each time "*the Suns*" first, and "*Phoenix*" subsequently. The treatment of proper nouns must thus also extend to such compound cases.

Now consider the object of the losing team's post-modifying clause. Using the nominal-internal syntactic roles of SURGE-1.0 it could be analyzed as follows:

(1)	"their	third	home	game	in a row"
	Determiner	*Ordinal*	*Classifier*	*Head*	*Qualifier*

Semantically this analysis is erroneous because the PP "*in a row*" does not modify the head noun "*game*" like the PP "*against Phoenix*" does. This can be shown by the contrast between:
the perfectly fine (2) "*a home game against Phoenix*"
and the non-sensical (3) * "*a home game in a row*"
.

Instead, "*in a row*" really modifies the ordinal "*third*" as in the synonymous:
(4) "*their [third straight] home game*",
a fact obscured in (1) due to the discontinuous nature of the modification. Neither ordinal modification nor any type of nominal-rank discontinuous modification was covered in SURGE-1.0.

Another class of nominals not covered by SURGE-1.0 yet pervasive in quantitative domains is illustrated by Barkley's scoring statistic "*a season high 37 points*" in sentence (0).

The bracketing: "*[a] [season high] [37] [points]*" where "*season high*" and "*37*" are simply analyzed as a classifier and cardinal of the head "*points*" is invalid since the cardinal must always precede the classifier in an NP.

Similarly, the bracketing: "*[a] [[season high] 37] [points]*" where "*season high 37*" is analyzed as a complex cardinal (similarly to the "*third straight*" complex ordinal above) is also invalid since a *indefinite* determiner like "'*a*" cannot co-occur with a cardinal in an NP.

The only possible bracketing is thus: "*[a] [season high] [37 points]*", requiring the coverage of a new type of quantitative NP head called a *measure*. Such a specialized quantitative category must also be allowed to appear as classifier as shown by the game result NP
"*a 107-102 victory ...*".

The last syntactic construct of the sentence above not covered by SURGE-1.0 are partitives such as Majerle's shooting performance in sentence (0): "*6 of 8 three point shots*".

In conclusion, generating sentences like sentence (0) requires extending:

- The set of semantic relations, syntactic categories and matrix constituents of adverbial complements.
- The set of clausal moods.

- The set of syntactic categories appearing as nominal head, ordinal, classifier, and qualifier.
- The set of compound syntactic categories.

B.3.2 Extensions to the clause sub-grammar

In this section, I present the extensions I made to the clause sub-grammar of SURGE. The goal of these extensions is to expand the wide-coverage of SURGE from the single clause rank to the complex sentence rank. It thus focuses on the subpart of the transitivity system defining the various adverbial constituents of the clause. Among the various adverbial meanings it further focuses on those realizable by either a PP, a nominal or a subordinate clause. Adverbial meanings which are realizable only by way of an adverb were thus excluded from this study (since one of the recurrent theme of this thesis is cross-ranking paraphrasing to which semantic classes realizable by a single syntactic category have no import).

There were several sources for candidate adverbial complement classes to cover in SURGE-2.0:

- The sub-corpus of human-written summaries whose in-depth analysis was presented in section 2.
- The comprehensive lists of adverbial classes independently presented from different angles in Chapters 8, 9, 14 and 15 of [Quirk *et al.* 1985].
- The more limited lists of adverbial classes presented in [Halliday 1985], [Thompson and Longacre 1985] and [Talmy 1985].

In general, I considered only forms that occurred independently in two of these sources, considering each chapter of [Quirk *et al.* 1985] as an independent source. In what follows, I do not present the painstaking derivation and cross-examination process that this study involved. I instead presents the end result from the perspective of a SURGE user switching from the 1.0 to the 2.0 version. The linguistic issues and concepts related either to the adverbial classes entirely new to the 2.0 version or to the new treatment of classes partially covered the 1.0 version are discussed in some detail.

B.3.2.1 Syntactic types of adverbial constituents

There is an asymmetry between the treatment of participants and the treatment of circumstantials in SURGE-1.0. Participants are *thematic* roles that are mapped onto *syntactic* roles such as *subject, verb, direct-object.* These syntactic roles in turn appear in the `pattern` meta-attributes indicating to the linearizer the syntactic constituent structure of the sentence to generate. In contrast, circumstantials are *thematic* roles that are not mapped on any syntactic roles. The linearizer knows which should appear in the sentence string and where, because the pattern features contains paths that points directly to the circumstantial *thematic* roles themselves. So for example, the top-level pattern for the sentence "*Orlando defeated Toronto yesterday*" where "*yesterday*" is a *Time* circumstantial is:

```
(pattern
 (dots {synt-roles subject} dots {synt-roles verb} dots {synt-roles direct-object} dots
{circum time}))
```

This approach assumes that there is no syntactic distinction among the various circumstantials. However, [Quirk *et al.* 1985] distinguish between several classes of circumstantials (that they call adverbials), each characterized by a distinct syntactic behavior. First, they distinguish among *adjuncts*, *disjuncts*, *subjuncts* and *conjuncts.* Then, they further distinguish between *predicate adjuncts* and *sentence adjuncts.*

B.3.2.1.1 Adjuncts vs. Disjuncts Subjuncts and disjuncts primarily convey modality and discourse cues and can only be realized by adverbs (*e.g.,* "really", "only", "then"), and never by NPs, PPs or clauses. They are thus beyond the scope of the present extensions and are were not implemented in SURGE-2.0. In contrast, SURGE-2.0 incorporates the distinction between adjuncts and disjuncts. [Quirk *et al.* 1985] give four tests to distinguish adjuncts from disjuncts[21]. Only adjuncts can:

[21] And in fact from subjuncts and conjuncts as well.

- Be clefted.
- Appear in an alternative question.
- Appear within the scope of a focusing subjunct like "*only*".
- Be elicited by a question.

I illustrates these four test on the following pair of example sentences:
(1) "Bo missed the game **because he was injured**."
(2) "**Since he was injured**, Bo missed the game."

It is interesting to note that these two sentences are synonymous and that both contain a *Reason* adverbial (in bold). However, the dependent clause "*he was injured*" functions as an *adjunct* when linked to the matrix clause by the subordinating conjunction "*because*", but it functions as a *disjunct* when linked to the matrix by the subordinating conjunction "*since*" as shown by the four following tests:

- *Clefting*:
  - "It was because he was injured that Bo missed the game."
  - "? It was since he was injured that Bo missed the game."
- *Alternative question*:
  - "Did Bo miss the game because he was injured or because he was suspended?"
  - "? Did Bo miss the game since he was injured or since he was suspended?"
- *Focusing subjunct*:
  - "Did Bo miss the game only because he was injured?"
  - "? Did Bo miss the game only since he was injured?"
- *Elicited by question*:
  - "Why did Bo miss the game? Because he was injured."
  - "? Why did Bo miss the game? Since he was injured."

The example above underlines the importance of making systematic distinctions among adverbials: the decision by SURGE to map a *Reason* circumstantial specified in its input onto an *adjunct* or *disjunct* is both lexically and syntactically constrained.

B.3.2.1.2 Predicate vs. Sentence Adjuncts Predicate adjuncts are distinguished from sentence adjuncts on both semantic and distributional grounds. Semantically, a predicate adjunct modifies only the verb of the clause whereas a sentence adjunct modifies the whole clause. This can be probed by using the question form "*What X did to Y was to Z*" as illustrated below. Predicate adjuncts are thus more nuclear to the clause than sentence adjuncts. In fact, as shown in Fig. B.20, the clause can be analyzed as a set of successive layers, from the verb, which is at the core, to disjuncts at the outer frontier, with the intermediate syntactic functions in between.

Distributionally, predicate adjuncts cannot appear in frontal position without the intention to produce a striking rhetorical effect appropriate only in very restricted contexts. In contrast, sentence adjunct can be freely fronted without affecting the rhetorical effect of the sentence. Moreover, when both a predicate adjunct and a sentence adjunct co-occur in trailing position in the same clause, the predicate adjunct must appear first. The two sentences below illustrate the difference between predicate and sentence adjuncts:

(3)	She kissed Bo **on the cheek**	(predicate adjunct)
(4)	She kissed Bo **on the platform**	(sentence adjunct)

The contrast between the predicate adjunct of (3) and the sentence adjunct of (4) is underlined by the following test transformation sentences.

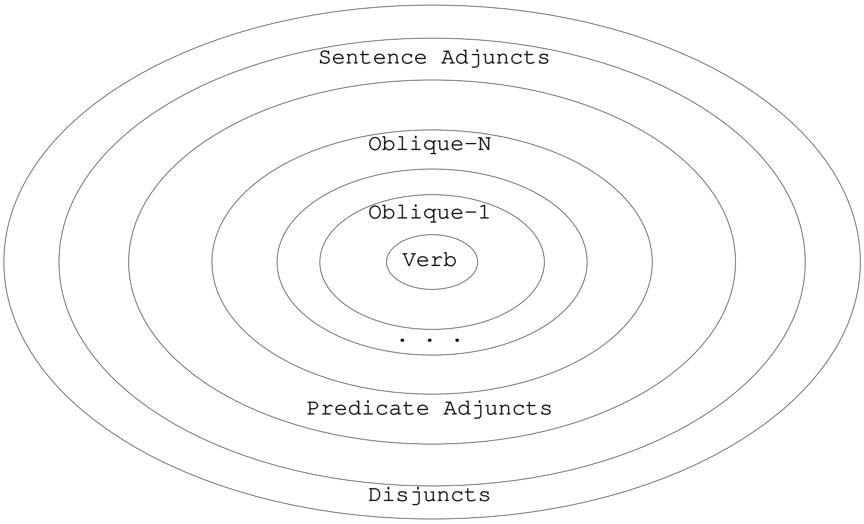

Figure B.20: Nuclearity of clause constituents

First, rhetorically,
(3b) “On the cheek, she kissed Bo.” $\neq$ “She kissed Bo on the cheek”
whereas
(4b) “On the platform, she kissed Bo.” $\Leftrightarrow$ “She kissed Bo on the platform”

Then,
(3c) “What she did to Bo was to kiss him on the cheek”
(3d) ? “What she did to Bo on the cheek was to kiss him”
whereas
(4c) ? “What she did to Bo was to kiss him on the platform”
(4d) “What she did to Bo on the platform was to kiss him”

Furthermore,
(5a) “On the platform, she kissed Bo on the cheek”
(5b) ? “On the cheek, she kissed Bo on the platform”
and
(5c) “She kissed Bo on the cheek on the platform”
(5d) ? “She kissed Bo on the platform on the cheek”

Sentences (3-4) above show that adverbials of the same semantic class, in the case at hand *location*, can be realized by either a predicate or a sentence adjunct depending on their precise content. Sentences (5a-c) show the even more interesting fact that two adverbials with the same semantics can co-occur in the same sentence as long as one fills a predicate adjunct function and the other fills a sentence adjunct function. That is (5a) is analyzed as:

On the platform,	she	kissed	Bo	on the cheek
Location	*Agent*	*Process*	*Affected*	*Location*
Sentence Adjunct	*Subject*	*Verb*	*Object*	*Predicate Adjunct*

The precise realization and co-occurrence constraints of each adverbial semantic class are discussed in the next section which review in turn each class implemented in SURGE-2.0. I now explain the general scheme for the input representation and syntactic realization of adverbials in SURGE-2.0. The distinction between

```
(
 ;; Input specification of ''She kissed Bo on the cheek on the platform''.
 (cat clause)
 (process ((type material) (lex "kiss")))
 (partic ((agent ((cat personal-pronoun) (gender feminine)))
          (affected ((cat proper) (lex "Bo")))))
 (pred-modif ((location ((cat pp)
                         (prep ((lex "on")))
                         (np ((cat common) (lex "cheek")))))))
 (circum ((location ((cat pp)
                     (prep ((lex "on")))
                     (np ((cat common) (lex "platform")))))))

;; SURGE-2.0 enrichments encoding the mapping to syntactic roles.
 (oblique ((1 {partic agent})
           (2 {partic affected})))
 (synt-roles ((verb {process})
              (subject {oblique 1})
              (subject ((synt-funct subject)))
              (direct-object {oblique 2})
              (direct-object ((synt-funct object)))
              (end-adverbial-1 {pred-modif location})
              (end-adverbial-1 ((synt-funct pred-adjunct)))
              (end-adverbial-2 {circum location})
              (end-adverbial-2 ((synt-funct sent-adjunct)))))
 (pattern
  (dots {synt-roles subject} dots {synt-roles verb} dots {synt-roles direct-object}
   dots {synt-roles end-adverbial-1} {synt-roles end-adverbial-2} dots))
 )
```

Figure B.21: Thematic to syntactic role mapping for adverbials in SURGE-2.0

disjuncts and adjuncts is purely syntactic and is thus internal to SURGE. However, since the distinction between predicate and sentence adjunct is also semantic and depends on the particular meaning of the open-class words contained in the adverbial (*e.g.,* "*cheek*" vs. "*platform*" in the example above), it must be encoded in the input to SURGE. Thus, in addition to the top-level features **process**, **partic** and **circum** of a SURGE-1.0 input, a SURGE-2.0 input also contains a **pred-modif** (abbreviation for predicate-modifier) feature whose elements are mapped onto *predicate adjuncts*. The elements under **circum** are in contrast mapped onto either *sentence adjuncts* or *disjuncts* depending on a combination of semantic, lexical and syntactic constraints.

The SURGE-2.0 input for sentence (5a) above is shown at the top of Fig. B.21. Note that with SURGE-1.,0, since all adverbials were appearing under **circum** in the input, and since an FD cannot contain twice the same attribute at a given level, a sentence with two location roles like (5a) could not have been generated. The way SURGE-2.0 maps the four types of thematic roles it accepts in input - process, participants, predicate-modifiers and circumstantials - onto syntactic roles is shown at the bottom of the same figure. The mapping of the process and participants remains unchanged from SURGE-1.0. However, while SURGE-1.0 did *not* map circumstantials onto any syntactic role, SURGE-2.0 *does* map both predicate-modifiers and circumstantials on a new class of adverbial syntactic roles.

Each of the obligatory syntactic roles that realize participants has a distinct syntactic behavior. In systemic linguistic terms, they fulfill a different syntactic function (abbreviated **synt-funct**): *Subject, Verb* etc. In contrast, there are only three syntactic functions for adverbial syntactic roles, *Predicate-Adjunct, Sentence-Adjunct* and *Disjunct*, but several adverbial constituent with the same function can co-occur in a given sentence. Therefore, instead of being distinguished by syntactic functions as for the obligatory syntactic roles, adverbial syntactic roles are distinguished in term of their position in the linear clause pattern.

In the current implementation, there are seven different positions available for an adverbial constituent: two frontal positions before the verb and any of its arguments and five trailing positions after the verb and all its arguments. In English, a variety of medial positions, located between two elements of the verb argument structure, are also available, albeit in much more restricted ways than the frontal and trailing positions. Deciding on the appropriateness of placing a particular adverbial in one of these medial position involves a complex combination of semantic, syntactic and stylistic factors. Since in most sentences, all adverbials appear in either frontal or trailing positions, I have left study of the complex constraints on medial position placing for future work.

To map adverbial thematic roles onto syntactic roles, SURGE-2.0 operates in two stages:

1. Taking into account the semantics, syntax and lexical content of the input thematic role, choose a syntactic function.
2. Taking into account the chosen syntactic function and the other adverbial constituents already mapped onto syntactic roles, choose one of the adverbial positions.

All predicate-modifiers are mapped onto predicate-adjuncts. For most circumstantials, the choice between a sentence-adjunct and a disjunct is entirely pre-determined on semantic grounds. For example, the *Time* thematic role can only be mapped onto a sentence adjunct and the *Condition* thematic role can only be mapped onto a disjunct. For others, it depends on syntactic constraints. For example, the *Manner* thematic role must be mapped onto a sentence adjunct when filled by an adverb and onto a disjunct when filled by a PP. This choice can also depend on lexical constraints. For example, a subordinated clause filling the *Reason* thematic role must be mapped onto a sentence adjunct when linked to the matrix clause by the subordination conjunction "*because*", but onto a disjunct when linked to the matrix clause by the subordination conjunction "*since*". Since SURGE works bi-directionally, either side of such constraints can be specified in the input and the opposing side will be enforced in the output by unification.

Once the syntactic function of each adverbial has been chosen, SURGE-2.0 then maps each of them onto specific slots in the precedence pattern of the clause in the following order: predicate adjuncts first, then sentence adjuncts, then disjuncts. For a predicate adjunct only the trailing slots are available and the leftmost available one is taken. For a sentence adjunct or a disjunct, five trailing and two frontal slots are available. The innermost available one is chosen, where the innermost order is defined as follows:
end-1 front-1 end-2 end-3 end-4 end-5 front-2.

This default placement can be overrode by explicitly restricting a given circumstantial to either frontal or trailing position in the input. Each adverbial thematic role present in the input specification is processed by the algorithm above in the following order:
For predicate modifiers: *Score, Manner, Instrument, Comparison, Matter, Distance, Location, Path, Origin, Destination, Duration,*
and for circumstantials: *Manner, Accompaniment, Opposition, Matter, Standard, Perspective, Behalf, Distance, Location, Duration, Frequency, Time, Means, Reason, Purpose, Addition, Comparison, Concession, Contrast, Exception, Substitution, Condition, Concessive-Condition, Inclusion, Co-Event, Result.*

Each of these thematic roles is defined and exemplified in the next subsection.

B.3.2.2 Extending the coverage of adverbial thematic roles

The extended coverage of adverbial thematic roles and their syntactic realization provided by SURGE-2.0 is given in figures B.22, B.23 and B.24. The first of these three figures contains the thematic roles realizable by predicate adjuncts, the second contains those realizable by sentence adjuncts and the third contains those realizable by disjuncts. Each of these figure is a table with one line for each syntactic realization available for each semantic type of adverbial covered. The semantic type is specified first in terms of a general

semantic			syntactic	Example	#	
class	role	feature	category			
Locative	Location		Adverb	Bo kissed her **there**.	1	
			PP	Bo kissed her **on the cheek**.	2	
			finite S	Bo kissed her **where she wanted**.	3	
			verbless S	Bo kept the keys **where convenient**.	4	
	Direction		Adverb	Bo slid **down**.	5	
			NP	Bo drove **this way**.	6	
			PP	Bo ran **up the hill**.	7	
	Destination		Adverb	Bo went **home**.	8	
			PP	Bo sent the letter **to LA**.	9	*
			finite S	She sent Bo back **where he belongs**.	10	
	Path		PP	Bo traveled **via Denver**.	11	
	Distance		NP	Bo came **a long way**.	12	
Temporal	Duration		Adverb	Bo stayed there **forever**.	13	
			NP	Bo stayed there **a long time**.	14	
Process	Manner		PP	Bo kiss her **with love**.	15	
	Means		Adverb	Bo was treated **surgically**.	16	
			PP	Bo was treated **by surgery**.	17	
	Instrument	polar+	PP	Bo pushed him **with both hands**.	18	*
		polar-	PP	Bo negotiated **without an agent**.	19	
	Comparison		finite S	Bo went there, **as he did yesterday**.	20	
			infinitive S	Bo played hard, **as if to send the fans a message**.	21	
			-ing S	Bo played great, **as if peeking on schedule**.	22	
			-ed S	Bo played hard, **as if not bothered by his knee**.	23	
			verbless S	Bo played great, **as if in great form**.	24	
Respect	Matter		PP	Bo talked **about his contract**.	25	
Domain	Score		Score	New York beat Indiana **90-87**.	26	

Figure B.22: Predicate Adjuncts in SURGE-2.0

class of semantic relation, then a specific thematic role belonging to that class and finally any finer-grained distinction encoded as a semantic feature appearing inside the role. An example sentence is given for each possible realization, with the sentence constituent corresponding to the input thematic role in bold.

The $<$ semantic-type , syntactic realization $>$ pairs that were already covered by SURGE-1.0 are indicated by a star in the last column of the table. Quantitatively, the extensions increased the number of such pairs covered by SURGE from 16 to 94. Qualitatively, the extensions acted in four different ways. First, they cover entirely new semantic classes of adverbials. For example, there was no conditional thematic role available in SURGE-1.0. Second, they added new members of partially covered classes. For example, the *Addition*, *Opposition*, *Exception*, *Inclusion* and *Substitution* thematic roles were added to the determinative class of adverbials which initially consisted of only one role: *Accompaniment*. Third, they completed the set of syntactic realizations of thematic roles that were only realizable by one or two syntactic categories in SURGE-1.0. For example, whereas locations could be expressed only as PPs, they can now also be expressed as adverbs, finite clauses and verbless clauses. Fourth, they created a comprehensive framework in the light of which the few ad-hoc treatments that SURGE-1.0 contained were revealed and re-thought, namely the locative and temporal adverbials as discussed in the paragraph dedicated to them in what follows.

There are eight semantic classes of adverbials in SURGE-2.0: locative, temporal, causative, conditional, process, determinative, respect, and domain-specific. I review the thematic roles of each of these classes in turn in the following paragraph.

semantic			syntactic	Example	#	
class	role	feature	category			
Locative	Location		Adverb	**There**, Bo kissed her.	1	
			PP	**On the platform**, Bo kissed her.	2	*
	Origin		PP	**From Kansas City**, Bo called Gwen.	3	*
	Distance		PP	**For a few miles**, the road is damaged.	4	
Temporal	Time		Adverb	**Yesterday**, Bo triumphed.	5	*
			NP	**Last year**, Bo triumphed.	6	
			PP	**On Monday**, Bo triumphed.	7	*
			finite S	**As he received the ball**, Bo smiled.	8	*
			-ing S	**After receiving the ball**, Bo smiled.	9	*
			-ed S	**Once injured**, Bo grimaced.	10	*
			verbless S	**Once on the floor**, Bo grimaced.	11	
	Duration		PP	**For a long time**, Bo stayed here.	12	
			finite S	**Until Bo recovered**, they waited for him.	13	
			-ing S	**Since joining the Raiders**, Bo shines.	14	
			-ed S	**Until forcibly removed**, they will stay.	15	
			verbless S	**As long as necessary**, they will stay here.	16	
	Frequency		Adverb	**Often**, Bo scored twice.	17	
			NP	**Twice a week**, Bo runs 10 miles.	18	
			PP	**On Sundays**, Bo run 10 miles.	19	
Causative	Reason		PP	**Because of injury**, Bo did not play.	20	*
			finite S	**Because he was injured**, Bo did not play.	21	*
	Purpose		PP	**For enough money**, Bo would play anywhere.	22	*
			finite S	**So he would get a raise**, Bo held out.	23	
			infinitive S	**To gain free-agency**, Bo held out.	24	*
	Behalf		PP	**For the Raiders,** Bo scored twice.	24	*
Determinative	Addition		-ing S	**In addition to trading Bo**, they waived Mike.	25	
	Accomp-	polar+	PP	**With her**, Bo would go anywhere.	26	*
	-animent	polar-		**Without her**, Bo wouldn't go anywhere.	27	
	Opposition		PP	**Against the Knicks**, the Nets are 3-1.	28	
Process	Manner		Adverb	**Tenderly**, Bo kissed her.	29	*
	Means		-ing S	**By acquiring Bo**, they strengthen their defense.	30	
	Comparison	polar+	PP	**Like Mike**, Bo soared above the rim.	31	
		polar-	PP	**Unlike Mike**, Bo can bat.	32	

Figure B.23: Sentence Adjuncts in SURGE-2.0

B.3.2.2.1 Locative adverbials There are six locative adverbial roles in SURGE-2.0: *Location, Direction, Destination, Origin, Path and Distance.* They respectively answer the questions, where?, in what direction?, to where?, from where?, through/across where? and how far?

Syntactically, they can surface as predicate adjuncts, sentence adjuncts or both. They can all co-occur in the same clause as in:

"In France,	Bo biked	south	200 miles	along the Rhone	from Lyon	to the Camargue."
Location		*Direction*	*Distance*	*Path*	*Origin*	*Destination*

The distinction among these SURGE-2.0 locative roles here is semantic as opposed to the lexically based distinction used in SURGE-1.0. Characterizing locative thematic roles by spatial prepositions is problematic for two reasons:

- Most locative meaning have alternative, non-prepositional realizations.
- There are many spatial prepositions in English and their respective meanings overlap in a complex, context sensitive ways [Herskovits 1986].

semantic			syntactic	Example	#
class	role	feature	category		
Temporal Causative Blend	Co-Event	habit+	finite S	**Whenever you hurt**, call me.	1
		habit-	-ing S	**With his knees hampering him**, Bo's defense is sloppy.	2
			ed S	**Injured against the Giants**, Bo didn't play.	3
			verbless S	**Unable to play due to injury**, Bo stayed home.	4
		habit+	verbless S	**Whenever in doubt**, call me.	5
Causative	Reason		finite S	**Since he was injured**, Bo did not play.	6
	Result		finite S	They wouldn't give him a raise, **so Bo held out**.	7
			infinitive S	They waived Bo, **only to see him flourish elsewhere**.	8
Conditional	Condition	polar+	finite S	**If he is fully fit**, Bo will play.	9
			-ed S	**If well-conditioned**, Bo will play.	10
			verbless S	**If in great shape**, Bo will play.	11
		polar-	finite S	**Unless he is fully fit**, Bo won't play.	12
			-ed S	**Unless well-conditioned**, Bo won't play.	13
			verbless S	**Unless in great shape**, Bo won't play.	14
	Concessive Condition		finite S	**Even if he is fully fit**, Bo won't play.	15
			-ed S	**Even if well-conditioned**, Bo won't play.	16
			verbless S	**Even if in great shape**, Bo won't play.	17
	Concession		PP	**In spite of his injury**, Bo played.	18
			finite S	**Although he was injured**, Bo played.	19
			-ed S	**Although injured**, Bo played.	20
			-ing S	**Although hurting**, Bo played.	21
			verbless S	**Although out of shape**, Bo played.	22
Determinative	Contrast		PP	**As opposed to many others**, Bo never held out.	23
			finite S	**Whereas Mike keeps shooting**, Bo prefers passing.	24
	Exception		PP	**Except Laettner**, all Olympian were pros.	25
			-ing S	**Except for blocking shots**, Bo can do it all.	26
	Inclusion		PP	Bo scored 30 points, **including 5 three-pointers**.	27
			-ing S	Bo can do it all, **including blocking shots**.	28
			verbless S	Bo played everywhere, **including in New York**.	29
	Substitution		PP	**Instead of Mike**, they picked Bo.	30
			-ing S	**Rather than drafting Mike**, they picked Bo.	31
			infinitive S (bare)	**Rather than shoot**, Bo made the perfect pass.	32
	Addition		PP	They picked Bo, **as well as Mike**.	33
Respect	Matter		PP	**Concerning Bo**, they were wrong.	34
	Standard		PP	**For a center**, Bo is very quick.	35
	Perspective		PP	**As a scorer**, Bo remains prolific.	36

Figure B.24: Disjuncts in SURGE-2.0

It is thus far preferable to define thematic roles by conceptual classes of meanings, independently of any specific realization or lexical item, using co-occurrence inside the same clause, as a differentiating criteria. The *At-Loc, In-Loc* and *On-Loc* roles of SURGE-1.0 all correspond to the PP realization of the single *Location* role of SURGE-2.0. The *From-Loc* and *To-Loc* roles of SURGE-1.0 respectively correspond the PP realization of the *Origin* and *Destination* roles in SURGE-2.0. SURGE-1.0's interpretation of NPs as PPs in the input specification of adverbials was possible only because it was not covering any NP realization of adverbials. SURGE-2.0 still chooses a default preposition for PP realizations of adverbials, but the category specified in the input for a PP must be PP and not an NP.

B.3.2.2.2 Temporal adverbials There are three temporal adverbial roles in SURGE-2.0: *Time, Duration* and *Frequency*. They respectively answer the questions, when?, how long? and how often? They can all co-occur in the same clause as in:

"Bo works out	four hours	every day	this month."
	Duration	*Frequency*	*Time*

The *Time* role of SURGE-2.0 reifies the *Time* and *Temporal-Background* roles of SURGE-1.0 whose sole distinction appeared to be that the former were realized by adverbs and PPs and the latter by clauses.

B.3.2.2.3 Causative adverbials There are four causative adverbials in SURGE-2.0: *Reason, Purpose, Result* and *Behalf* who all answer the questions *why?* and individually answer the questions *for what reason? for what purpose? with what result? on whose behalf?*. These four thematic roles are mutually exclusive. *Result* can only be realized by a clause and only appear in trailing position. *Behalf* can only be realized by a PP whose NP complement is animate. *Reason* and *Purpose* can both be realized by either a clause or a PP. They can be easily distinguished by the different subordination conjunctions or prepositions used to link their content to the matrix clause: *"since"* or *"because"* and *"because of"* for *Reason*, *"in order to"* or *"so that"* and *"for"* for *Purpose*.

B.3.2.2.4 Co-Event clauses The *Co-Event* adverbial forms a semantic class on its own, since it is used to link two events related by an unspecified relation that blends causative, temporal and sometimes even locative elements. It can only be only realized by clauses, though by a variety of clausal forms. It can only fill the disjunct syntactic function. I distinguish between two types of Co-Events, habitual ones, as in:
(1) "Whenever there is smoke, there is fire."
and regular ones, as in:
(2) "Injured against the Giants, Bo did not play."

Habitual Co-Event clauses[22] express a recurrent pattern of co-occurrence (whether effective or desired) between the event they convey and the event conveyed in the matrix clause. They can be either finite or verbless.

In contrast, regular Co-Event clauses[23] express a isolated co-occurrence between the event they convey and the event conveyed in the matrix clause. They can be either participial or verbless.

The fact that Co-Event clauses are an unspecified causative-temporal blend is well illustrated by the following examples:
(1a) "Before there is smoke, there is fire."
(1b) "There is smoke because there is fire."
(2a) "After he got injured against the Giants, Bo did not play."
(2b) "Because he got injured against the Giants, Bo did not play."

(1) above can be equally well interpreted as synonymous to the temporal (1a) and the causative (1b). Similarly, (2) above can be equally well interpreted as synonymous to the temporal (2a) and the causative

[22] Called "contingency" clauses in [Quirk *et al.* 1985], pp:1086.
[23] Called "absolutive" clauses by [Thompson and Longacre 1985] pp:200-201 and "suppletive" clauses by [Quirk *et al.* 1985], pp:1120-27.

(2b). Their versatility and conciseness make Co-Event clauses a construct of choice in the corpora of human-written summaries I analyzed.

B.3.2.2.5 Conditional adverbials There are three conditional adverbials in SURGE-2.0: *Condition, Concessive-Condition* and *Concession*, and they can surface only as disjuncts. *Condition* and *Concessive-Condition* can only be realized by clauses. *Concession* can also be realized by a PP. The polarity feature distinguishes between positive conditions, whose default subordinator is *"if"* and negative conditions, whose default subordinator is *"unless"*. Further subdivisions of condition clauses, such as the *predictive vs. hypothetical vs. counter-factual* distinctions discussed in [Thompson and Longacre 1985] interacts tightly with the modality system of the grammar and were thus left for a future round of extension.

B.3.2.2.6 Process adverbials There are four "process" adverbials in SURGE-2.0: *Manner, Means, Instrument* and *Comparison.* They all answer the question *how?* and specify in some way the nature of the process by which the event described in the matrix clause occurred. They are all adjuncts. Depending on the syntactic category that realizes them, the *Manner, Means* and *Comparison* thematic roles can be either predicate or sentence adjuncts. *Instrument* can only be realized by a predicate adjunct PP. All four process adverbials co-occur in the same clause, as in:

Shrewdly,	Bo went home	by train	with his free-ticket	like he did yesterday.
Manner		*Means*	*Instrument*	*Comparison*

B.3.2.2.7 Respect adverbials There are three respect adverbials in SURGE-2.0: *Matter, Standard* and *Perspective*, all realizable only by PPs. *Matter* specifies the topic which the proposition conveyed by the matrix clauses must be interpreted as referring to. *Standard* specifies a comparative reference frame for evaluating the proposition conveyed by the matrix clause. *Perspective* specifies the particular role or function of the entity mentioned in the matrix clause that is relevant for the interpretation of the proposition conveyed by that clause. These three adverbials are mutually exclusive in a given clause.

B.3.2.2.8 Determinative adverbials There are seven determinative adverbials in SURGE-2.0: *Accompaniment, Addition, Inclusion, Substitution, Exception, Opposition* and *Contrast.* Although several of them are described by different authors, there seem to be very little agreement with respect to which semantic class of adverbial meanings each of these thematic roles belongs to. I decided to group all of them under the "determinative" label because, although they form a semantically diverse set, they all function at the clause rank in a manner similar to a determiner at the nominal rank: they circumscribe the participants of the matrix clauses in the context of a larger set of domain entities. Determinative adverbials can be realized by either PPs or clauses, filling either a sentence adjunct or disjunct function.

B.3.2.2.9 Domain-specific adverbials Finally, SURGE-2.0 incorporates the *Score* adverbial, which is specific to the particular sports domain of STREAK. This example illustrates that even if comprehensive and thorough, a general, portable set of thematic roles may still require minor extensions when applied to a new application domain.

B.3.2.3 Moods of subordinated adverbial clauses

The clausal realizations of the thematic roles of SURGE-2.0 includes three types of clauses not covered in SURGE-1.0:

- Bound adverbial clauses
- Verbless clauses
- Bare infinitive clauses

The mood system thus need to be extended as shown in Fig. B.25.

The mood *bound* designates finite subordinate clauses. Inside the matrix clause, these subordinate clauses can fill either a nominal function (*i.e., Subject, Object*) or an adverbial one (*i.e., Adjunct, Disjunct*). As shown in Fig. B.2, SURGE-1.0 covered only the nominal variety of bound clauses and the non-finite variety of adverbial clauses. In bound adverbial clauses, the subject is often *controlled* by either the subject or object of the matrix clause. In this context, "controlled" means omitted but understood as co-referent. Controlled subject in bound clauses are similar to scoped subject in relative clauses (cf. section B.2.4). In SURGE-2.0, they are handled by a path valued feature `control` indicating which thematic role in the subordinate clause is to be syntactically omitted. The description of this thematic role reduces to a path pointing to the controlling thematic role in the matrix. When SURGE sees such paths while performing subject-verb agreement, it then knows that the subject must be gaped and, by following the path chain, which matrix syntactic role contains the relevant features for the agreement. An example input of bound adverbial clause with controlled subject is given in Fig. B.26. All the features relevant to agreement are centralized under the `index` features.

The reason for leaving the subject implicit in dependent clauses is to achieve conciseness. There is an even more concise form of dependent clauses, called *verbless* clauses, in which the verb itself is omitted. Such clauses can in general be equally well interpreted as realizing different process types. For example the verbless clause in italics in the table of Fig. B.25 can be equally well interpreted as an abbreviated form of the clause:
(1) "*Although playing without Ewing*"
or of the clause:
(2) "*Although they were without Ewing*"

In (1) the process type is *material* and "*without Ewing*" is an accompaniment adjunct. In (2) the process type is *locative* and "*without Ewing*" is an accompaniment complement. Since each verbless clause has a valid relational interpretation, I treat them as such is SURGE-2.0.

The third new mood of SURGE-2.0 is *bare-infinitive*, the variety of infinitive clauses that can be directly attached to the matrix clauses without the need for the subordinating conjunction "*to*".

finite	bound	adverbial	Ewing scored 28 points *as the Knicks won the game*
non-finite	verbless		*Although without Ewing*, the Knicks won the game
	infinitive	bare	Ewing helped the Knicks *win the game*

Figure B.25: Extensions to the mood system

```
;; ‘‘Dan Majerle made 6 of 8 three point shots to add 24’’
((cat clause)
 (tense past)
 (process ((type material) (effect-type creative) (lex "make")))
 (partic ((agent ((cat person-name) (first-name ((lex "Dan"))) (last-name "Majerle")))
          (created ((cat partitive)
                    (part ((cat cardinal) (value 6) (digit yes)))
                    (part-of ((cat common)
                              (cardinal ((cat cardinal) (value 8) (digit yes)))
                              (classifier ((cat measure)
                                           (quantity ((cat cardinal) (value 3)))
                                           (unit ((cat noun) (lex "point")))))
                              (head ((cat noun) (lex "shot"))))))))
 (circum ((result ((cat clause)
                   (mood to-infinitive)
                   (process ((type material) (effect-type creative) (lex "add")))

                   (controlled {^ partic agent})
                   (partic ((agent ((index {^5 partic agent index})))

                            (created ((cat measure) (value 24) (unit ((gap yes)))))))))))
```

Figure B.26: Example input description with controlled subject

B.3.3 Extensions to the nominal sub-grammar

The extensions to the nominal sub-grammar from SURGE-1.0 to SURGE-2.0 were sizeable and are summed-up in the table of Fig. B.27. They were carried out in two distinct ways: definition of new nominal sub-categories and definition of new modification relations among existing categories.

The three new categories are: *measure, noun-compound* and *partitive*. Measure is a category specially designed for statistics. It has two attributes: *quantity*, whose value must be a cardinal and *unit* whose value can be either a noun or a noun-compound. Measure can appear both as a head or as a classifier. Noun-compound handles deep cases of noun-noun modifications. It can appear anywhere in place of a noun. It is has two attributes: *classifier* and *head*. Each can recursively be a noun-compound. Even though classifier and head are two attributes of the full NP, this noun-compound category is needed, because these two feature cannot appear without a determiner in full a NP[24]. The partitive category handles fractional measures and proportions. It has two attributes *part* whose value can be a cardinal, an ordinal, a measure or a full NP and *part-of* whose value can be either a measure or a full NP. It can appear anywhere instead of a measure. The input specification of a complex nominal example containing subconstituents of the three categories just defined in given in Fig. B.28. Figure B.26 contains another example input of partitive constituent.

Two existing categories that were inherently atomic in SURGE-1.0 accept modifiers in SURGE-2.0: ordinals and proper nouns. Ordinals can be modified either by adjectives (cf. first row in Fig. B.27), or, discontinually, by PPs (cf. second row in Fig. B.27). Proper nouns can be modified by the same range of modifiers than common nouns. In addition, they can also have their own internal compound structure, which in a way parallels the new noun-compound category introduced for common nouns. These two new properties of proper nouns are illustrated by the example below:

"The	surging	New York	Knicks	who have won three straight"
Determiner	*Describer*	*Head*		*Qualifier*
Article	*Participle*	*Compound Proper*		*Relative Clause*
		Home	*Franchise*	

B.3.4 Extensions to the rest of the grammar

In SURGE-1.0, paratactic complexes could not be modified in any way. SURGE-2.0 allows conjunctions (or appositions) of clauses to be modified by adverbials. The adverbial is attached using a **`pred-modif`** or **`circum`** feature exactly as in the case of a simple clause. A sub-grammar for dates and one for addresses were also incorporated to SURGE-2.0.

B.3.5 Summary

SURGE-2.0 extends the wide coverage of SURGE-1.0 at the simple clause rank, both down to the nominal-rank and up to the complex sentence rank. The extensions have been based both on corpus data and descriptive linguistic works. They are comprehensive and insure the robustness of SURGE-2.0 for any quantitative domain, even those including the most complex and concise constructions of English. Beyond STREAK, SURGE-2.0 has been used for developing generation applications in three other domains, and I expect it to soon become the "official" version of SURGE for distribution of the FUF/SURGE generation environment package.

[24] Even though plural count NPs may have an empty determiner, cf. Fig. B.13.

grammatical function	syntactic category	example
Ordinal	Numeral phrase	their **third straight** victory
	discontinuous Numeral phrase	their **third** victory **in a row**
Classifier	Noun compound	a **franchise record** victory
	Measure	a **33 point** performance
Head	proper Noun	the hapless **Denver Nuggets** who lost again
	Noun compound	his league **season high**
	Measure	a career high **33 points**
	Partitive	a season best **12 of 16 shots**
Qualifier	present-participle S	a victory **invigorating the Knicks**
	past-participle S	a victory **hard-fought until the end**

Figure B.27: Extensions to the nominal grammar

```
;; ``a franchise record 60.5 percent of their three point field goal attempts''
((cat common)
 (definite no)
 (classifier ((cat noun-compound)
              (classifier ((lex "franchise")))
              (head ((lex "record")))))
 (head ((cat partitive)
        (part ((cat measure)
               (quantity ((value 60.5)))
               (unit ((lex "percent")))))
        (part-of ((cat common)
                  (number plural)
                  (determiner ((cat personal-pronoun)))
                  (classifier ((cat noun-compound)
                               (classifier ((cat measure)
                                            (quantity ((value 3) (digit no)))
                                            (unit ((lex "point")))))
                               (noun ((cat noun-compound)
                                      (classifier ((lex "field")))
                                      (head ((lex "goal")))))))
                  (head ((lex "attempt"))))))))
```

Figure B.28: Input description of a complex nominal in SURGE-2.0

B.4 Limitations of SURGE 2.0

The wide coverage of SURGE-2.0 at the four main linguistic ranks: determiner, nominal, simple clause and complex sentence makes it the most comprehensive portable syntactic front-end for generation application in English available today. It remains, however, incomplete and requires further extension in a variety of ways. In the following paragraph, I overview the most needed extensions in each major system of the grammar.

B.4.1 Transitivity

As explained in Section B.2.2.1 there are three main classes of clauses: those expressing an event, those expressing a relation and those, called composite, expressing both, with the event as cause and the relation as consequent. There are also three main classes of events: material, mental and verbal. Although all three types can take part in a composite clause, only composite clauses with a *material* event component are covered in the current implementation. The first needed extension of the transitivity system is to cover composite clauses with *mental* and *verbal* event components as well.

The hierarchy of general process types defining the deep argument structure of a clause (and the semantic class of its main verb) in the current implementation is a synthesis from [Halliday 1985], [Fawcett 1987] and [Lyons 1977]. This hierarchy is compact and able to cover many clause structures. Yet the argument structure and/or semantics of many English verbs do not fit neatly in any element of this hierarchy. This is why the lexical processes have been added to SURGE as an alternative form of input specification for clause structures. However, lexical processes are a shallower and less semantic form of input than the general process types. For example, the sub-categorization constraints and the mapping from the thematic roles to the oblique roles have to be specified in the input instead of being automatically computed by the grammar. Minimizing the use of this lexicalist approach to infrequent and authentically idiosyncratic clause structures would be highly desirable. It would require both refining and extending the current hierarchy of general process types. An excellent starting point for carrying out this task is the comprehensive taxonomy of English verbs proposed by [Levin 1993].

B.4.2 Adverbials

The coverage of the adverbial system in the current implementation is limited in two ways. First, among the four types of syntactic functions that adverbial constituents can fill in a clause – adjuncts, disjuncts, conjuncts and subjuncts – only the first two are covered. Second, the only adverbial semantic roles covered are those which can be realized by either a clausal, prepositional or nominal form. But for some adverbial semantic roles, the sole possible realization is an adverb. Those semantic roles are not currently covered. These two limitations are correlated: most conjuncts (*e.g., alternatively, correspondingly*) or subjuncts (*e.g., politically, cordially*) are adverbs and they convey semantics roles with no alternative syntactic realizations.

The grammar provides a default ordering of clause constituents realizing adverbial semantic roles. This default specifies for a given set of input roles, which will appear in frontal position (*i.e.,* before the verb and any of its arguments) and which will appear in trailing positions (*i.e,* after the verb and all of its arguments). Another limitation of the current adverbial system is that, though it allows input specifications overriding the default choice of frontal *vs.* trailing position for a given role, it does not allow input specifications overriding the default *relative* ordering among several frontal or trailing adverbials. Also, the current grammar considers only the frontal or trailing position and not the many medial positions available in clauses with multiple argument verbs and/or multi-word verb groups. This last ordering limitation is related to the semantic and syntactic limitations mentioned above: medial positions are very rarely occupied by the clausal, prepositional or nominal adjuncts and disjuncts. They are in contrast, the position of choice for adverbs functioning as conjuncts and subjuncts.

B.4.3 Mood

Some forms of subordinate clauses are not yet covered by the current mood system. This is the case for example of the forms below (where the subordinate clause is highlighted in bold):

- "We must find out **how to handle mood in this framework**."
- "We must discover **whether we can handle mood in this framework**."
- "We must know **what they did**."

Currently, the mood of a clause is encoded as a single feature whose values form a complex hierarchy. The multi-dimensionality of the mood variations would be better captured by switching to a multi-feature representation based on the set of mostly orthogonal and binary features: `finite, imperative, question, question-type, subordinate, subordinate-type, embedded, subjunctive`. With such a set, the mood of a subordinate clause such as:
"The customer persuaded the programmer **that there is a bug**"
which is currently encoded as: `(mood bound-nominal-declarative)`
would become encoded as:

```
(mood ((finite yes)
       (subordinate yes)
       (subordinate-type bound-nominal)
       (subjunctive no)))
```

Another necessary improvement to the mood system is to extend the currently quite limited set of factors that influence both the choices of binder and the restrictive status in relative clauses.

B.4.4 Voice

In terms of structural alternations affecting the focus and ordering of information in the clause, the current grammar covers only the passive and dative transformations. The voice system thus needs to be extended to include the following alternations as well:

- Clefting, *e.g.,* "Bo runs" *vs.* "It is Bo who runs"
- Argument fronting[25], *e.g.,* "Bo broke his personal record yesterday" *vs.* "His personal record, Bo broke yesterday"
- It-extraposition, *e.g.,* "Whether Bo scores or not won't matter" *vs.* "It won't matter whether Bo scores or not".

B.4.5 Nominals

The most urgent improvement of the nominal system is to extend it to cover reflexive pronouns. Another improvement would consist of refining the set of nominal functions currently available, namely `determiner, describer, classifier, head` and `qualifier`. In particular, [Fries 1970] proposes a more complete set of functions. It also includes detailed constraints on co-occurrence of several elements filling the same function (with different semantics) in a given NP. Finally, the work of [Vendler 1968] and [Levi 1978] on nominalizations and non-predicative adjectives could serve as a good – though quite limited in scope – starting point, towards the development of a set of thematic roles for a semantic input specification of nominals paralleling the set of thematic roles already used for the semantic input specification of clauses.

[25] Distinct from *adverbial* fronting which the current grammar already handles.

B.5 Coverage of SURGE-2.0

SURGE comes with a large set of test inputs that to be run after each change to the grammar. This input text set systematically probes each branch of the grammar and many combinations of features from different branches. The initial input set was incrementally created over seven years during the development of the generation systems COMET [McKeown *et al.* 1990] [McKeown *et al.* 1993], COOK [Smadja and McKeown 1991] and ADVISOR-II [Elhadad 1993b]. The test inputs of SURGE-1.0 are presented in [Elhadad 1993b]. In this section, I present the extension of the input set testing the extensions from from SURGE-1.0 to SURGE-2.0. This additional test input set provides a detailed account of the extended coverage of SURGE-2.0.

B.5.1 Test inputs for the extensions at the nominal rank

Simple and complex nominals with quantities

```
(def-test t300
  "52 points."
  ((cat measure)
   (quantity ((value 52)))
   (unit ((lex "point")))))

(def-test t301
  "Five rebounds."
  ((cat measure)
   (quantity ((value 5)))
   (unit ((lex "rebound")))))

(def-test t302
  "5 blocked shots."
  ((cat measure)
   (quantity ((value 5) (digit yes)))
   (unit ((lex "blocked shot")))))

(def-test t303
  "Six 3 point shots."
  ((cat common)
   (cardinal ((value 6)))
   (definite no)
   (classifier ((cat measure)
                (quantity ((value 3) (digit yes)))
                (unit ((lex "point")))))
   (head ((lex "shot")))))

(def-test t304
  "Season high."
  ((cat noun-compound)
   (classifier ((lex "season")))
   (head ((lex "high")))))

(def-test t305
  "The pleasant house property tax office furniture."
  ((cat common)
   (describer ((lex "pleasant")))
   (classifier ((cat noun-compound)
```

```
             (classifier ((cat noun-compound)
                          (classifier ((cat noun-compound)
                                       (classifier ((lex "house")))
                                       (head ((lex "property")))))
                          (head ((lex "tax")))))
             (head ((lex "office")))))
   (head ((lex "furniture")))))

(def-test t306
  "A season high 27 points."
  ((cat common)
   (definite no)
   (classifier ((cat noun-compound)
                (classifier ((lex "season")))
                (head ((lex "high")))))
   (head ((cat measure)
          (number plural)
          (quantity ((value 27)))
          (unit ((lex "point")))))))

(def-test t307
  "Stockton's season high 27 points."
  ((cat common)
   (possessor ((cat basic-proper)
               (lex "Stockton")))
   (classifier ((cat noun-compound)
                (classifier ((lex "season")))
                (head ((lex "high")))))
   (head ((cat measure)
          (quantity ((value 27)))
          (unit ((lex "point")))))))

(def-test t308
  "Fourth quarter."
  ((cat measure)
   (quantity ((cat ordinal)
              (value 4)))
   (unit ((lex "quarter")))))

(def-test t309
  "17 fourth quarter points."
  ((cat measure)
   (quantity ((value 17)))
   (unit ((cat noun-compound)
          (classifier ((cat measure)
                       (quantity ((cat ordinal)
                                  (value 4)))
                       (unit ((lex "quarter")))))
          (head ((lex "point")))))))

(def-test t310
  "A playoff record six fourth quarter three pointers."
  ((cat common)
   (definite no)
```

```
   (classifier ((cat noun-compound)
                (classifier ((lex "playoff")))
                (head ((lex "record")))))
   (head ((cat measure)
          (quantity ((value 6)))
          (unit ((cat noun-compound)
                 (classifier ((cat measure)
                              (quantity ((cat ordinal) (value 4)))
                              (unit ((lex "quarter")))))
                 (head ((lex "three pointer")))))))))

(def-test t311
  "The Kings's NBA record 35th straight road loss."
  ((cat common)
   (possessor ((cat basic-proper)
               (lex "the Kings")))
   (classifier ((cat noun-compound)
                (classifier ((lex "NBA")))
                (head ((lex "record")))))
   (head ((cat measure)
          (quantity ((cat ordinal) (value 35)))
          (unit ((cat noun-compound)
                 (classifier ((cat list)
                              (distinct ~(((cat adj)
                                           (lex "straight"))
                                          ((cat noun)
                                           (lex "road"))))))
                 (head ((lex "loss")))))))))

(def-test t312
  "The Hawks, winners of six straight games and five in a row at the Omni."
  ((cat np)
   (complex apposition)
   (distinct
    ~(((cat basic-proper)
       (number plural)
       (lex  "Hawk"))
      ((cat common)
       (definite no)
       (number plural)
       (head ((lex "winner")))
       (qualifier
        ((cat pp)
         (prep ((lex "of")))
         (np ((cat np)
              (complex conjunction)
              (distinct
               ~(((cat common)
                  (definite no)
                  (cardinal ((value 6)))
                  (classifier ((cat adj) (lex "straight")))
                  (head ((lex "game"))))
                 ((cat common)
                  (definite no)
```

```
                  (cardinal ((value 5)))
                  (head ((gap yes)))
                  (qualifier ((cat list)
                              (distinct
                               ~(((cat pp)
                                  (prep ((lex "in")))
                                  (np ((cat common)
                                       (definite no)
                                       (head ((lex "row"))))))
                                 ((cat pp)
                                  (prep ((lex "at")))
                                  (np ((cat common)
                                       (lex "Omni"))))))))))))))))))))

(def-test t313
  "13 of 14 shots."
  ((cat partitive)
   (part ((value 13)))
   (part-of ((cat common)
             (definite no)
             (cardinal ((value 14)))
             (head ((lex "shot")))))))

(def-test t314
  "13 of 14 shots."
  ((cat partitive)
   (part ((value 13)))
   (part-of ((cat measure)
             (quantity ((value 14)))
             (unit ((lex "shot")))))))

(store-plurals '(("percent" "percent")))

(def-test t315
  "60.5 percent of their field goal attempts."
  ((cat partitive)
   (part ((cat measure)
          (quantity ((value 60.5)))
          (unit ((lex "percent")))))
   (part-of ((cat common)
             (possessor ((cat personal-pronoun)
                         (person third)
                         (number plural)))
             (classifier ((lex "field goal")))
             (number plural)
             (head ((lex "attempt")))))))

(def-test t316
  "60.5 percent of their field goal attempts."
  ((cat partitive)
   (part ((cat common)
          (definite no)
          (cardinal ((value 60.5)))
          (head ((lex "percent")))))
```

```
   (part-of ((cat common)
             (possessor ((cat personal-pronoun)
                         (person third)
                         (number plural)))
             (classifier ((lex "field goal")))
             (number plural)
             (head ((lex "attempt")))))))

(def-test t317
  "His NBA season high."
  ((cat common)
   (possessor ((cat personal-pronoun)
               (person third)
               (gender masculine)
               (number singular)))
   (classifier ((lex "NBA")))
   (head ((cat noun-compound)
          (classifier ((lex "season")))
          (head ((lex "high")))))))

(def-test t318
  "His NBA season high performance of 52 points."
  ((cat common)
   (possessor ((cat personal-pronoun)
               (person third)
               (gender masculine)
               (number singular)))
   (classifier ((cat noun-compound)
                (classifier ((lex "NBA")))
                (head ((cat noun-compound)
                       (classifier ((lex "season")))
                       (head ((lex "high")))))))
   (head ((lex "performance")))
   (qualifier ((cat pp)
               (prep ((lex "of")))
               (np ((cat common)
                    (definite no)
                    (cardinal ((value 52)))
                    (head ((lex "point")))))))))

(def-test t319
  "His NBA season high performance of 52 points."
  ((cat partitive)
   (part ((cat common)
          (possessor ((cat personal-pronoun)
                      (person third)
                      (gender masculine)
                      (number singular)))
          (classifier ((cat noun-compound)
                       (classifier ((lex "NBA")))
                       (head ((cat noun-compound)
                              (classifier ((lex "season")))
                              (head ((lex "high")))))))
          (head ((lex "performance")))))
```

```
   (part-of ((cat measure)
             (quantity ((value 52)))
             (unit ((lex "point")))))))

(def-test t320
  "6 of an NBA team record 22 blocked shots."
  ((cat partitive)
   (part ((value 6) (digit yes)))
   (part-of ((cat common)
             (definite no)
             (classifier ((cat noun-compound)
                          (classifier ((lex "NBA") (a-an an)))
                          (head ((cat noun-compound)
                                 (classifier ((lex "team")))
                                 (head ((lex "record")))))))
             (head ((cat measure)
                    (quantity ((value 22)))
                    (unit ((lex "blocked shot")))))))))

(def-test t321
  "A team record 47 of 53 free throws."
  ((cat common)
   (definite no)
   (classifier ((cat noun-compound)
                (classifier ((lex "team")))
                (head ((lex "record")))))
   (head ((cat partitive)
          (part ((value 47)))
          (part-of ((cat measure)
                    (quantity ((value 53)))
                    (unit ((lex "free throw")))))))))

(def-test t322
  "1 percent lowfat."
  ((cat ap)
   (classifier ((cat measure)
                (quantity ((value 1) (digit yes)))
                (unit ((lex "percent")))))
   (head ((lex "lowfat")))))

(def-test t323
  "Two heaping soup spoons of grade A 1 percent lowfat pasteurized milk."
  ((cat partitive)
   (part ((cat measure)
          (quantity ((value 2)))
          (unit ((cat noun-compound)
                 (classifier ((cat verb)
                              (ending present-participle)
                              (lex "heap")))
                 (head ((cat noun-compound)
                        (classifier ((lex "soup")))
                        (head ((lex "spoon")))))))))
   (part-of ((cat common)
             (countable no)
```

```
             (definite no)
             (describer ((cat list)
                         (distinct ~(((cat basic-proper)
                                      (lex "grade A"))
                                     ((cat ap)
                                      (classifier ((cat measure)
                                                   (quantity ((value 1)
                                                              (digit yes)))
                                                   (unit ((lex "percent")))))
                                      (head ((lex "lowfat"))))
                                     ((cat verb)
                                      (ending past-participle)
                                      (lex "pasteurize"))))))
             (head ((lex "milk")))))))

(def-test t324
  "All 28 of its free throws."
  ((cat partitive)
   (total +)
   (part ((value 28)))
   (part-of ((cat common)
             (possessor ((cat personal-pronoun)
                         (person third)
                         (number singular)
                         (gender neuter)))
             (head ((lex "free throw")))))))

(def-test t325a
  "3 point range."
  ((cat common)
   (determiner none)
   (classifier ((cat measure)
                (quantity ((value 3)
                           (digit  yes)))
                (unit ((lex "point")))))
   (head ((lex "range")))))

(def-test t325
  "Five of five from 3 point range."
  ((cat partitive)
   (part ((value 5)))
   (part-of ((cat common)
             (definite no)
             (cardinal ((value 5)))
             (head ((gap yes)))
             (qualifier ((cat pp)
                         (prep ((lex "from")))
                         (np ((cat common)
                              (determiner none)
                              (classifier ((cat measure)
                                           (quantity ((value 3)
                                                      (digit  yes)))
                                           (unit ((lex "point")))))
                              (head ((lex "range"))))))))))
```

```
(def-test t326
  "A perfect 12 for 12 from the line."
  ((cat common)
   (definite no)
   (describer ((cat adj) (lex "perfect")))
   (head ((cat partitive)
          (prep ((lex "for")))
          (part ((value 12)))
          (part-of ((cat common)
                    (definite no)
                    (cardinal ((value 12)))
                    (head ((gap yes)))
                    (qualifier ((cat pp)
                                (prep ((lex "from")))
                                (np ((cat common)
                                     (head ((lex "line")))))))))))))

(def-test t327
  "Stockton scored 27 points."
  ((cat clause)
   (tense past)
   (process ((type material)
             (effect-type creative)
             (lex "score")))
   (partic ((agent ((cat basic-proper)
                    (lex "Stockton")))
            (created ((cat measure)
                      (quantity ((value 27)))
                      (unit ((lex "point")))))))))


(def-test t328
  "Stockton scored a season high 27 points."
  ((cat clause)
   (tense past)
   (process ((type material)
             (effect-type creative)
             (lex "score")))
   (partic ((agent ((cat basic-proper)
                    (lex "Stockton")))
            (created ((cat common)
                      (definite no)
                      (classifier ((cat noun-compound)
                                   (classifier ((lex "season")))
                                   (head ((lex "high")))))
                      (head ((cat measure)
                             (quantity ((value 27)))
                             (unit ((lex "point"))))))))))))


(def-test t329
  "Ewing scored 25 of his 29 points in the first half."
  ((cat clause)
```

```
   (tense past)
   (process ((type material)
             (effect-type creative)
             (lex "score")))
   (partic ((agent ((cat basic-proper)
                    (lex "Ewing")))
            (created ((cat partitive)
                      (part ((value 25)))
                      (part-of ((cat common)
                                (possessor ((cat personal-pronoun)
                                            (person third)
                                            (gender masculine)
                                            (number singular)))
                                (head ((cat measure)
                                       (quantity ((value 29)))
                                       (unit ((lex "point"))))))))))
   (circum ((in-loc ((cat common)
                     (ordinal ((value 1)))
                     (head ((lex "half")))))))))


(def-test t357
  "John Stockton scored 27 points."
  ((cat clause)
   (tense past)
   (process ((type material)
             (effect-type creative)
             (lex "score")))
   (partic ((agent ((cat compound-proper)
                    (head ((cat person-name)
                           (first-name ((lex "John")))
                           (last-name ((lex "Stockton")))))))
            (created ((cat measure)
                      (quantity ((value 27)))
                      (unit ((lex "point")))))))))

(def-test t358
  "John Stockton scored a season high 27 points."
  ((cat clause)
   (tense past)
   (process ((type material)
             (effect-type creative)
             (lex "score")))
   (partic ((agent ((cat compound-proper)
                    (head ((cat person-name)
                           (first-name ((lex "John")))
                           (last-name ((lex "Stockton")))))))
            (created ((cat common)
                      (definite no)
                      (classifier ((cat noun-compound)
                                   (classifier ((lex "season")))
                                   (head ((lex "high")))))
                      (head ((cat measure)
                             (quantity ((value 27)))
```

```
                            (unit ((lex "point"))))))))))

(def-test t359
  "John Stockton's 27 points."
  ((cat common)
   (possessor ((cat compound-proper)
               (head ((cat person-name)
                      (first-name ((lex "John")))
                      (last-name ((lex "Stockton")))))))
   (head ((cat measure)
          (quantity ((value 27)))
          (unit ((lex "point")))))))

(def-test t360
  "John Stockton's season high 27 points."
  ((cat common)
   (possessor ((cat compound-proper)
               (head ((cat person-name)
                      (first-name ((lex "John")))
                      (last-name ((lex "Stockton")))))))
   (classifier ((cat noun-compound)
                (classifier ((lex "season")))
                (head ((lex "high")))))
   (head ((cat measure)
          (quantity ((value 27)))
          (unit ((lex "point")))))))

(def-test t361
  "The Sacramento Kings' NBA record 35th straight road loss."
  ((cat common)
   (possessor ((cat compound-proper)
               (number plural)
               (head ((cat team-name)
                      (home ((lex "Sacramento")))
                      (franchise ((lex "King")))))))
   (classifier ((cat noun-compound)
                (classifier ((lex "NBA")))
                (head ((lex "record")))))
   (head ((cat measure)
          (quantity ((cat ordinal) (value 35)))
          (unit ((cat noun-compound)
                 (classifier ((cat list)
                              (distinct ~(((cat adj)
                                           (lex "straight"))
                                          ((cat noun)
                                           (lex "road"))))))
                 (head ((lex "loss")))))))))
```

Simple and complex nominals with proper names

```
(def-test t331
  "Johnson."
  ((cat person-name)
   (last-name ((lex "Johnson")))))

(def-test t332
  "Earvin."
  ((cat person-name)
   (first-name ((lex "Earvin")))))

(def-test t333
  "Magic."
  ((cat person-name)
   (nickname ((lex "Magic")))))

(def-test t334
  "Earvin Johnson."
  ((cat person-name)
   (first-name ((lex "Earvin")))
   (last-name ((lex "Johnson")))))

(def-test t335
  "Magic Johnson."
  ((cat person-name)
   (nickname ((lex "Magic")))
   (last-name ((lex "Johnson")))))

(def-test t336
  "Earvin Magic Johnson."
  ((cat person-name)
   (first-name ((lex "Earvin")))
   (nickname ((lex "Magic")))
   (last-name ((lex "Johnson")))))

(def-test t337
  "Rufus T. Firefly."
  ((cat person-name)
   (first-name ((lex "Rufus")))
   (middle-name ((lex "T.")))
   (last-name ((lex "Firefly")))))

(def-test t338
  "Ramses II."
  ((cat person-name)
   (first-name ((lex "Ramses")))
   (dynasty ((cat ordinal) (value 2)))))

(def-test t339
  "Earvin Johnson Sr."
  ((cat person-name)
   (dynasty ((father yes)))
   (first-name ((lex "Earvin")))
```

```
   (last-name ((lex "Johnson")))))

(def-test t340
  "Earvin Johnson Jr."
  ((cat person-name)
   (dynasty ((father no)))
   (first-name ((lex "Earvin")))
   (last-name ((lex "Johnson")))))

(def-test t341
  "James Baker III."
  ((cat person-name)
   (first-name ((lex "James")))
   (last-name ((lex "Baker")))
   (dynasty ((cat ordinal) (value 3)))))

(def-test t342
  "Hugo Z. Hackenbush XXIII."
  ((cat person-name)
   (first-name ((lex "Hugo")))
   (middle-name ((lex "Z.")))
   (dynasty ((cat ordinal) (value 23)))
   (last-name ((lex "Hackenbush")))))

(def-test t343
  "Dr. Elhadad."
  ((cat person-name)
   (title ((lex "Dr.")))
   (last-name ((lex "Elhadad")))))

(def-test t344
  "Mr. Bulent Fishkin."
  ((cat person-name)
   (title ((lex "Mr.")))
   (first-name ((lex "Bulent")))
   (last-name ((lex "Fishkin")))))

(def-test t345
  "Dr. Rufus T. Firefly."
  ((cat person-name)
   (title ((lex "Dr.")))
   (first-name ((lex "Rufus")))
   (middle-name ((lex "T.")))
   (last-name ((lex "Firefly")))))

(def-test t346
  "Prof. Hugo Z. Hackenbush XXIII."
  ((cat person-name)
   (title ((lex "Prof.")))
   (first-name ((lex "Hugo")))
   (middle-name ((lex "Z.")))
   (dynasty ((cat ordinal) (value 23)))
   (last-name ((lex "Hackenbush")))))
```

```
(def-test t347a
  "Doctor Marshall President."
  ((cat list)
   (distinct ~(((lex "Doctor") (cat noun))
               ((lex "Marshall") (cat noun))
               ((lex "President") (cat noun))))))

(def-test t347
  "Doctor Marshall President Idi Amin Dada."
  ((cat person-name)
   (title   ((cat list)
             (distinct ~(((lex "Doctor") (cat noun))
                         ((lex "Marshall") (cat noun))
                         ((lex "President") (cat noun))))))
   (first-name ((lex "Idi")))
   (middle-name ((lex "Amin")))
   (last-name ((lex "Dada")))))

(def-test t348
  "Dr. Rufus T. Firefly and Dr. Hugo Z. Hackenbush."
  ((cat np)
   (complex conjunction)
   (distinct ~(((cat compound-proper)
                (head ((cat person-name)
                       (title ((lex "Dr.")))
                       (first-name ((lex "Rufus")))
                       (middle-name ((lex "T.")))
                       (last-name ((lex "Firefly"))))))
               ((cat compound-proper)
                (head ((cat person-name)
                       (title ((lex "Dr.")))
                       (first-name ((lex "Hugo")))
                       (middle-name ((lex "Z.")))
                       (last-name ((lex "Hackenbush")))))))))

(def-test t349
  "Mr. and Ms. Fishkin."
  ((cat np)
   (complex conjunction)
   (distinct ~(((cat compound-proper)
                (head ((cat person-name)
                       (title ((lex "Mr.")))
                       (last-name ((gap yes))))))
               ((cat compound-proper)
                (head ((cat person-name)
                       (title ((lex "Ms.")))
                       (last-name ((lex "Fishkin")))))))))


(def-test t350
  "The great Serigne Suleyman Abdul Aziz Seck."
  ((cat compound-proper)
   (describer ((lex "great")))
   (head ((cat person-name)
```

```
          (title ((lex "Serigne")))
          (first-name ((lex "Suleyman")))
          (middle-name ((lex "Abdul Aziz")))
          (last-name ((lex "Seck")))))))

(def-test t351
  "Denver."
  ((cat team-name)
   (home ((lex "Denver")))))

(def-test t351b
  "Denver."
  ((cat compound-proper)
   (head ((cat team-name)
          (home ((lex "Denver")))))))

(def-test t351c
  "Denver."
  ((cat basic-proper)
   (lex "Denver")))

(def-test t352
  "The Nuggets."
  ((cat compound-proper)
   (number plural)
   (head ((cat team-name)
          (franchise ((lex "Nugget")))))))

(def-test t352b
  "The Nuggets."
  ((cat basic-proper)
   (number plural)
   (lex "Nugget")))

(def-test t353
  "The Denver Nuggets."
  ((cat compound-proper)
   (number plural)
   (head ((cat team-name)
          (home ((lex "Denver")))
          (franchise ((lex "Nugget")))))))

(def-test t354
  "The hapless Denver Nuggets."
  ((cat compound-proper)
   (number plural)
   (describer ((lex "hapless")))
   (head ((cat team-name)
          (home ((lex "Denver")))
          (franchise ((lex "Nugget")))))))

(def-test t355
  "The Denver Nuggets who extended their losing streak to five games."
  ((cat compound-proper)
```

```
   (number plural)
   (qualifier ((cat clause)
               (process ((type composite)
                         (relation-type locative)
                         (lex "extend")))
               (mood relative)
               (tense past)
               (scope {^ partic agent})
               (partic ((affected ((cat common)
                                   (possessor ((cat personal-pronoun)
                                               (person third)
                                               (number plural)))
                                   (classifier ((cat verb)
                                                (ending present-participle)
                                                (lex "lose")))
                                   (head ((lex "streak")))))
                        (located {^ affected})
                        (location ((cat pp)
                                   (prep ((lex "to")))
                                   (np ((cat measure)
                                        (quantity ((value 5)))
                                        (unit ((lex "game"))))))))))
   (head ((cat team-name)
          (home ((lex "Denver")))
          (franchise ((lex "Nugget")))))))

(def-test t356
  "The hapless Denver Nuggets who extended their losing streak to six games."
  ((cat compound-proper)
   (number plural)
   (describer ((lex "hapless")))
   (qualifier ((cat clause)
               (process ((type composite)
                         (relation-type locative)
                         (lex "extend")))
               (mood relative)
               (tense past)
               (scope {^ partic agent})
               (partic ((affected ((cat common)
                                   (possessor ((cat personal-pronoun)
                                               (person third)
                                               (number plural)))
                                   (classifier ((cat verb)
                                                (ending present-participle)
                                                (lex "lose")))
                                   (head ((lex "streak")))))
                        (located {^ affected})
                        (location ((cat pp)
                                   (prep ((lex "to")))
                                   (np ((cat measure)
                                        (quantity ((value 6)))
                                        (unit ((lex "game"))))))))))
   (head ((cat team-name)
          (home ((lex "Denver")))
```

```
          (franchise ((lex "Nugget")))))))


(def-test t362
  "The Atlanta Hawks, winners of six straight games and five in a row at the Omni."
  ((cat np)
   (complex apposition)
   (distinct
    ~(((cat compound-proper)
       (number plural)
       (head ((cat team-name)
              (home ((lex "Atlanta")))
              (franchise ((lex "Hawk"))))))
      ((cat common)
       (definite no)
       (number plural)
       (head ((lex "winner")))
       (qualifier
        ((cat pp)
         (prep ((lex "of")))
         (np ((cat np)
              (complex conjunction)
              (distinct
               ~(((cat common)
                  (definite no)
                  (cardinal ((value 6)))
                  (classifier ((cat adj) (lex "straight")))
                  (head ((lex "game"))))
                 ((cat common)
                  (definite no)
                  (cardinal ((value 5)))
                  (head ((gap yes)))
                  (qualifier ((cat list)
                              (distinct
                               ~(((cat pp)
                                  (prep ((lex "in")))
                                  (np ((cat common)
                                       (definite no)
                                       (head ((lex "row"))))))
                                 ((cat pp)
                                  (prep ((lex "at")))
                                  (np ((cat common)
                                       (lex "Omni"))))))))))))))))))))


(def-test t363
  "Dr. Rufus T. Firefly gave Prof. Hugo Z. Hackenbush his best regards."
  ((cat clause)
   (tense past)
   (process ((type composite)
             (relation-type possessive)
             (lex "give")))
   (partic ((agent ((cat compound-proper)
                    (head ((cat person-name)
```

```
                        (title ((lex "Dr.")))
                        (first-name ((lex "Rufus")))
                        (middle-name ((lex "T.")))
                        (last-name ((lex "Firefly")))))))
          (affected ((cat compound-proper)
                     (head ((cat person-name)
                            (title ((lex "Prof.")))
                            (first-name ((lex "Hugo")))
                            (middle-name ((lex "Z.")))
                            (last-name ((lex "Hackenbush")))))))
          (possessor {^ affected})
          (possessed ((cat common)
                      (number plural)
                      (possessor ((cat personal-pronoun)
                                  (person third)
                                  (number singular)
                                  (gender masculine)))
                      (describer ((lex "best")))
                      (head ((lex "regard")))))))))

(def-test t364
  "The highly respected Dr. Rufus T. Firefly gave Prof. Hugo Z. Hackenbush who
   received the Turing award yesterday his best regards."
  ((cat clause)
   (tense past)
   (process ((type composite)
             (relation-type possessive)
             (lex "give")))
   (partic
    ((agent ((cat compound-proper)
             (describer ((cat ap)
                         (modifier ((lex "highly")))
                         (head ((lex "respected")))))
             (head ((cat person-name)
                    (title ((lex "Dr.")))
                    (first-name ((lex "Rufus")))
                    (middle-name ((lex "T.")))
                    (last-name ((lex "Firefly")))))))
     (affected ((cat compound-proper)
                (head ((cat person-name)
                       (title ((lex "Prof.")))
                       (first-name ((lex "Hugo")))
                       (middle-name ((lex "Z.")))
                       (last-name ((lex "Hackenbush")))))
                (qualifier ((cat clause)
                            (tense past)
                            (mood relative)
                            (restrictive no)
                            (voice passive)
                            (process ((type composite)
                                      (agentive no)
                                      (relation-type possessive)
                                      (lex "receive")))
                            (scope {^ partic affected})
```

```
                            (partic
                             ((possessor {^ affected})
                              (possessed ((cat common)
                                          (classifier ((cat basic-proper)
                                                       (lex "Turing")))
                                          (head ((lex "award")))))))
                            (circum ((time ((cat adv)
                                            (lex "yesterday")))))))))
     (possessor {^ affected})
     (possessed ((cat common)
                 (number plural)
                 (possessor
                  ((cat personal-pronoun)
                   (semantics ((index {^4 agent semantics index})))))
                 (describer ((lex "best")))
                 (head ((lex "regard")))))))))

(def-test t364bis
  "The seemingly unstoppable San Antonio Spurs extended their winning streak to 30
   games with a 111 85 victory over the hapless Denver Nuggets, losers of 11 in a row."
  ((cat clause)
   (tense past)
   (process ((type composite)
             (relation-type locative)
             (lex "extend")))
   (partic
    ((agent ((cat compound-proper)
             (number plural)
             (describer ((cat ap)
                         (modifier ((lex "seemingly")))
                         (head ((lex "unstoppable")))))
             (head ((cat team-name)
                    (home ((lex "San Antonio")))
                    (franchise ((lex "Spur")))))))
     (affected ((cat common)
                (possessor ((cat personal-pronoun)
                            (number plural)
                            (person third)))
                (classifier ((cat verb)
                             (ending present-participle)
                             (lex "win")))
                (head ((lex "streak")))))
     (located {^ affected})
     (location ((cat pp)
                (prep ((lex "to")))
                (np ((cat measure)
                     (quantity ((value 30)))
                     (unit ((lex "game")))))))))
   (pred-modif
    ((instrument
      ((cat pp)
       (np ((cat common)
            (definite no)
            (classifier ((cat list)
```

```
                         (distinct ~(((cat measure)
                                       (quantity ((value 111)))
                                       (unit ((gap yes))))
                                      ((cat measure)
                                       (quantity ((value 85)))
                                       (unit ((gap yes))))))))
            (head ((lex "victory")))
            (qualifier
             ((cat pp)
              (prep ((lex "over")))
              (np ((cat np)
                   (complex apposition)
                   (distinct
                    ~(((cat compound-proper)
                       (number plural)
                       (describer ((lex "hapless")))
                       (head ((cat team-name)
                              (home ((lex "Denver")))
                              (franchise ((lex "Nugget"))))))
                      ((cat common)
                       (definite no)
                       (number plural)
                       (head ((lex "loser")))
                       (qualifier
                        ((cat pp)
                         (prep ((lex "of")))
                         (np ((cat common)
                              (definite no)
                              (cardinal ((value 11)))
                              (head ((gap yes)))
                              (qualifier
                               ((cat pp)
                                (prep ((lex "in")))
                                (np ((cat common)
                                     (definite no)
                                     (head ((lex "row"))))))))))))))))))))))))))

(def-test t364ter
  "The hapless Denver Nuggets, losers of 11 in a row."
  ((cat np)
   (complex apposition)
   (distinct
    ~(((cat compound-proper)
       (number plural)
       (describer ((lex "hapless")))
       (head ((cat team-name)
              (home ((lex "Denver")))
              (franchise ((lex "Nugget"))))))
      ((cat common)
       (definite no)
       (number plural)
       (head ((lex "loser")))
       (qualifier
        ((cat pp)
```

```
        (prep ((lex "of")))
        (np ((cat common)
             (definite no)
             (cardinal ((value 11)))
             (head ((gap yes)))
             (qualifier
              ((cat pp)
               (prep ((lex "in")))
               (np ((cat common)
                    (definite no)
                    (head ((lex "row"))))))))))))))))))


(def-test t364quad
  "Losers of 11 in a row."
  ((cat common)
   (definite no)
   (number plural)
   (head ((lex "loser")))
   (qualifier
    ((cat pp)
     (prep ((lex "of")))
     (np ((cat common)
          (definite no)
          (cardinal ((value 11)))
          (head ((gap yes)))
          (qualifier
           ((cat pp)
            (prep ((lex "in")))
            (np ((cat common)
                 (definite no)
                 (head ((lex "row"))))))))))))))


(def-test t365
  "San Antonio defeated Denver."
  ((cat clause)
   (tense past)
   (process ((lex "defeat")))
   (partic ((agent ((cat basic-proper)
                    (lex "San Antonio")))
            (affected ((cat basic-proper)
                       (lex "Denver")))))))

(def-test t365b
  "San Antonio defeated Denver."
  ((cat clause)
   (tense past)
   (process ((lex "defeat")))
   (partic ((agent ((cat compound-proper)
                    (head ((cat team-name)
                           (home ((lex "San Antonio")))))))
            (affected ((cat compound-proper)
                       (head ((cat team-name)
```

```
                              (home ((lex "Denver"))))))))))

(def-test t366
  "The Spurs defeated the Nuggets."
  ((cat clause)
   (tense past)
   (process ((lex "defeat")))
   (partic ((agent ((cat compound-proper)
                    (number plural)
                    (head ((cat team-name)
                           (franchise ((lex "Spur")))))))
            (affected ((cat compound-proper)
                       (number plural)
                       (head ((cat team-name)
                              (franchise ((lex "Nugget")))))))))))

(def-test t371
  "My advisor and friend, Steve."
  ((cat np)
   (complex apposition)
   (restrictive no)
   (distinct
    ~(((cat common)
       (head ((cat noun)
              (complex conjunction)
              (distinct ~(((lex "advisor"))
                          ((lex "friend"))))))
       (possessor ((cat personal-pronoun)
                   (person first))))
      ((cat basic-proper)
       (lex "Steve"))))))

(def-test t372
  "Three times the butter."
  ((cat common)
   (lex "butter")
   (countable no)
   (multiplier ((value 3)))
   (definite yes)))


(def-test t373
  "The man whose hat is on the ground."
  ((cat common)
   (lex "man")
   (animate yes)
   (qualifier ((cat clause)
               (mood relative)
               (process ((type locative)))
               (scope {^ partic located possessor})
               (partic ((located ((cat common) (lex "hat")))
                        (location ((cat pp)
                                   (prep ((lex "on")))
                                   (np ((cat common) (lex "ground"))))))))))
```

```
(def-test t374
  "Bowman, whose passing game is exceptional."
  ((cat compound-proper)
   (head ((cat person-name)
          (last-name ((lex "Bowman")))))
   (animate yes)
   (qualifier ((cat clause)
               (mood relative)
               (restrictive no)
               (process ((type ascriptive)))
               (scope {^ partic carrier possessor})
               (partic ((carrier ((cat common)
                                  (classifier ((cat verb)
                                               (ending present-participle)
                                               (lex "pass")))
                                  (head ((lex "game")))))
                        (attribute ((cat ap) (lex "exceptional")))))))))


(def-test t375
  "A 127 - 111 win over Denver sending the Nuggets to their seventh straight loss."
 ((cat common)
  (definite no)
  (classifier ((cat score)
               (win ((value 127)))
               (lose ((value 111)))))
  (head ((lex "win")))
  (qualifier
   ((cat list)
    (distinct ~(((cat pp)
                 (prep ((lex "over")))
                 (np ((cat compound-proper)
                      (head ((cat team-name)
                             (home ((lex "Denver"))))))))
                ((cat clause)
                 (mood present-participle)
                 (process ((type composite)
                           (relation-type locative)
                           (lex "send")))
                 (controlled {^ partic agent})
                 (partic ((agent ((index {^7 index})))
                          (located ((cat compound-proper)
                                    (number plural)
                                    (head ((cat team-name)
                                           (franchise ((lex "Nugget")))))))
                          (affected {^ located})
                          (location ((cat pp)
                                     (prep ((lex "to")))
                                     (np ((cat common)
                                          (possessor ((cat personal-pronoun)
                                                      (index {^4 located index})))
                                          (ordinal ((value 7)))
```

```
                                           (describer ((lex "straight")))
                                           (head ((lex "loss")))))))))))))))))


(def-test t376
  "John gives Mary his book."
  ((cat clause)
   (proc ((type composite) (relation-type possessive)))
   (partic ((agent ((cat basic-proper) (lex "John") (gender masculine)))
            (affected ((cat basic-proper) (lex "Mary") (gender feminine)))
            (possessor {^ affected})
            (possessed ((cat common)
                        (possessor ((cat personal-pronoun)
                                    (index  {^3 agent index})))
                        (head ((lex "book")))))))))
```

Dates

```
(def-test t387
  "Friday."
  ((cat date) (day-name ((lex "Friday")))))

(def-test t388
  "Friday night."
  ((cat date) (day-name ((lex "Friday"))) (day-part ((lex "night")))))

(def-test t389
  "June 1999."
  ((cat date) (month ((lex "June"))) (year ((value "1999")))))


(def-test t390
  "Friday the 13th, at night."
  ((cat date)
   (day-name ((lex "Friday")))
   (day-num ((value 13)))
   (day-part ((lex night)))))

(def-test t391
  "Friday 6 / 13 / 1999, in the evening."
  ((cat date)
   (day-name ((lex "Friday")))
   (day-num ((value 13)))
   (month ((value 6)))
   (year ((value 1999)))
   (day-part ((lex "evening")))))

(def-test t392
  ("Friday the 13th of June 1999, in the morning."
   "The morning of Friday June 13th 1999.")
  ((cat date)
   (day-name ((lex "Friday")))
   (day-num ((value 13)))
   (month ((lex "June")))
   (year ((value 1999)))
   (day-part ((lex "morning")))))


(def-test t394
  ("It happened the morning of Friday June 13th 1999."
   "It happened Friday the 13th of June 1999, in the morning.")
  ((cat clause)
   (tense past)
   (process ((type temporal) (lex "happen")))
   (partic ((located ((cat personal-pronoun)))
            (time ((cat date)
                   (day-name ((lex "Friday")))
                   (day-num ((value 13)))
                   (month ((lex "June")))
                   (year ((value 1999)))
```

```
                  (day-part ((lex "morning")))))))))


(def-test t395a
  "Saturday night -- Karl Malone scored 28 points with his hands."
  ((cat clause)
   (tense past)
   (process ((type material) (effect-type creative) (lex "score")))
   (partic ((agent ((cat compound-proper)
                    (gender masculine)
                    (head ((cat person-name)
                           (first-name ((lex "Karl")))
                           (last-name ((lex "Malone")))))))
            (created ((cat measure)
                      (quantity ((value 28)))
                      (unit ((lex "point")))))))
   (pred-modif ((instrument ((cat pp)
                             (np ((cat common)
                                  (number plural)
                                  (possessor ((cat personal-pronoun)
                                              (index {^5 partic agent index})))
                                  (head ((lex "hand")))))))))
   (circum ((time ((cat date)
                   (day-name ((lex "Saturday")))
                   (day-part ((lex "night")))
                   (position header)))))))

(def-test t395b
  "Karl Malone scored 28 points with his hands Saturday night."
  ((cat clause)
   (tense past)
   (process ((type material) (effect-type creative) (lex "score")))
   (partic ((agent ((cat compound-proper)
                    (gender masculine)
                    (head ((cat person-name)
                           (first-name ((lex "Karl")))
                           (last-name ((lex "Malone")))))))
            (created ((cat measure)
                      (quantity ((value 28)))
                      (unit ((lex "point")))))))
   (pred-modif ((instrument ((cat pp)
                             (np ((cat common)
                                  (number plural)
                                  (possessor ((cat personal-pronoun)
                                              (index {^5 partic agent index})))
                                  (head ((lex "hand")))))))))
   (circum ((time ((cat date)
                   (position end)
                   (day-name ((lex "Saturday")))
                   (day-part ((lex "night")))))))))
```

Addresses

```
(def-test t378
 "666 Visconde de Piraja Street, # 910, Ipanema."
  ((cat address)
   (num ((value 666)))
   (st-name ((lex "Visconde de Piraja")))
   (st-type ((lex "Street")))
   (apt-num ((lex "910")))
   (hood ((lex "Ipanema")))))

(def-test t379
  "666 West 112th Street, # 3D, Manhattan."
  ((cat address)
   (side ((lex "West")))
   (num ((value 666)))
   (st-name ((value 112)))
   (hood ((lex "Manhattan")))
   (apt-num ((lex "3D")))
   (st-type ((lex "Street")))))

(def-test t380
  "309 Michael Elhadad Boulevard, # 1244, N.W. Bersheva."
  ((cat address)
   (num ((value 309)))
   (st-name ((cat person-name)
             (first-name ((lex "Michael")))
             (last-name ((lex "Elhadad")))))
   (st-type ((lex "Boulevard")))
   (apt-num ((lex "1244")))
   (quadrant ((lex "N.W.")))
   (city ((lex "Bersheva")))))

(def-test t381
  "Pelourinho, Bahia, Brazil."
  ((cat address)
   (hood ((lex "Pelourinho")))
   (state ((lex "Bahia")))
   (country ((lex "Brazil")))))

(def-test t382
  "Rio de Janeiro, RJ, Brazil."
  ((cat address)
   (city ((lex "Rio de Janeiro")))
   (state ((lex "RJ")))
   (country ((lex "Brazil")))))

(def-test t383
  "666 Visconde de Piraja Street, # 910, Ipanema, Rio de Janeiro, RJ, Brazil."
  ((cat address)
   (num ((value 666)))
   (st-name ((lex "Visconde de Piraja")))
   (st-type ((lex "Street")))
   (apt-num ((lex "910")))
```

```
   (hood ((lex "Ipanema")))
   (city ((lex "Rio de Janeiro")))
   (state ((lex "RJ")))
   (country ((lex "Brazil")))))

(def-test t384
  "309 Michael Elhadad Boulevard, # 1244, N.W. Bersheva, The Negev, 84120 Israel."
  ((cat address)
   (num ((value 309)))
   (st-name ((cat person-name)
             (first-name ((lex "Michael")))
             (last-name ((lex "Elhadad")))))
   (st-type ((lex "Boulevard")))
   (apt-num ((lex "1244")))
   (quadrant ((lex "N.W.")))
   (city ((lex "Bersheva")))
   (state ((lex "The Negev")))
   (zip ((value 84120)))
   (country ((lex "Israel")))))

(def-test t385
  "544 Faidherbe Prolongee Avenue, Medina, P.O. Box 3275, Dakar, Senegal."
  ((cat address)
   (num ((value 544)))
   (st-name ((lex "Faidherbe Prolongee")))
   (st-type ((lex "Avenue")))
   (hood ((lex "Medina")))
   (po-box-num ((value 3275)))
   (city ((lex "Dakar")))
   (country ((lex "Senegal")))))

(def-test t386
  "San Antonio, Texas."
  ((cat address) (city ((lex "San Antonio"))) (state ((lex "Texas")))))


(def-test t393
  "I live 666 Visconde de Piraja Street, # 910, Ipanema, Rio de Janeiro, RJ, Brazil."
  ((cat clause)
   (process ((type locative) (lex "live")))
   (partic ((located ((cat personal-pronoun) (person first)))
            (location ((cat address)
                       (num ((value 666)))
                       (st-name ((lex "Visconde de Piraja")))
                       (st-type ((lex "Street")))
                       (apt-num ((lex "910")))
                       (hood ((lex "Ipanema")))
                       (city ((lex "Rio de Janeiro")))
                       (state ((lex "RJ")))
                       (country ((lex "Brazil")))))))))


(def-test t396
  "In Brazil, near Bahia."
```

```
  ((cat pp)
   (complex apposition)
   (restrictive no)
   (distinct ~(((prep ((lex "in")))
                (np ((cat basic-proper) (lex "Brazil"))))
               ((prep ((lex "near")))
                (np ((cat basic-proper) (lex "Bahia")))))))))


(def-test t397
  "In Bahia and in Rio."
  ((cat pp)
   (complex conjunction)
   (distinct ~(((prep ((lex "in")))
                (np ((cat basic-proper) (lex "Bahia"))))
               ((prep ((lex "in")))
                (np ((cat basic-proper) (lex "Rio")))))))))
```

B.5.2 Test inputs for the extensions at the complex sentence rank

```
;; Tests: location as pred-modif PP
;;        direction as adverb
;;        origin, destination and distance as PP
;;        path
;;        co-occurrence of all spatial roles
(def-test c1
  "In France, Bo biked south 300 miles along the Rhone from Lyon to the Camargue."
  ((cat clause)
   (tense past)
   (process ((type material) (effective no) (lex "bike")))
   (partic ((agent ((cat basic-proper) (lex "Bo")))))
   (pred-modif ((direction ((cat adv) (lex "south")))
                (distance ((cat measure)
                           (quantity ((value 300)))
                           (unit ((lex "mile")))))
                (path ((cat pp)
                       (prep ((lex "along")))
                       (np ((cat basic-proper) (lex "the Rhone")))))
                (origin ((cat pp)
                         (np ((cat basic-proper) (lex "Lyon")))))
                (destination ((cat pp)
                              (np ((cat basic-proper) (lex "the Camargue")))))))
   (circum ((location ((cat pp)
                       (prep ((lex "in")))
                       (np ((cat basic-proper)
                            (lex "France")))))))))


;; Tests: co-occurrence of 2 locations, 1 pred-modif & 1 circum
(def-test c2
  "On the platform, Bo kissed her on the cheek."
  ((cat clause)
   (tense past)
   (process ((type material) (lex "kiss")))
   (partic ((agent ((cat basic-proper) (lex "Bo")))
            (affected ((cat personal-pronoun) (gender feminine)))))
   (pred-modif ((location ((cat pp)
                           (prep ((lex "on")))
                           (np ((cat common) (lex "cheek")))))))
   (circum ((location ((cat pp)
                       (prep ((lex "on")))
                       (np ((cat common) (lex "platform")))))))))


;; Tests: location as adverb and finite clause
(def-test c3
  "There, Bo kissed her where she wanted."
  ((cat clause)
   (tense past)
   (process ((type material) (lex "kiss")))
   (partic ((agent ((cat basic-proper) (lex "Bo")))
```

```
          (affected ((cat personal-pronoun) (gender feminine)))))
   (pred-modif
    ((location ((cat clause)
                (mood bound-adverbial)
                (tense past)
                (process ((type mental) (lex "want") (transitive no)))
                (partic ((processor ((cat personal-pronoun)
                                     (index {^5 partic affected index})))))))))
   (circum ((location ((cat adv) (lex "there")))))))


;; Tests: duration as measure
;;        frequency as common
;;        time as common
;;        co-occurrence of 3 temporal roles
;;        inclusion as PP
(def-test c4
  "This month, Bo works out four hours each day, including Sundays."
  ((cat clause)
   (process ((type material) (effective no) (lex "work out")))
   (partic ((agent ((cat basic-proper) (lex "Bo")))))
   (pred-modif ((duration ((cat measure)
                           (quantity ((value 4)))
                           (unit ((lex "hour")))))))
   (circum ((frequency ((cat common)
                        (total +)
                        (number singular)
                        (lex "day")
                        (position end)))
            (time ((cat common) (distance near) (lex "month") (position front)))
            (inclusion ((cat pp)
                        (np ((cat common)
                             (number plural)
                             (definite no)
                             (lex "Sunday")))))))))

(store-verbs '(("work out" "works out" "worked out" "working out" "worked out")))


;; Tests: direction as common
;;        destination as adverb
;;        time as adverb
;;        frequency as adverb
(def-test c5
  "Often, Bo runs this way home now."
  ((cat clause)
   (process ((type material) (effective no) (lex "run")))
   (partic ((agent ((cat basic-proper) (lex "Bo")))))
   (pred-modif ((destination ((cat adv) (lex "home")))
                (direction ((cat common) (distance near) (lex "way")))))
   (circum ((frequency ((cat adv) (lex "often")))
            (time ((cat adv) (lex "now")))))))

(store-verbs '(("run" "runs" "ran" "running" "ran")))
```

```
(def-test c5bis
  "Bo runs home this way often now."
  ((cat clause)
   (process ((type material) (effective no) (lex "run")))
   (partic ((agent ((cat basic-proper) (lex "Bo")))))
   (end-adverbial-1 ((cat adv) (lex "home")))
   (end-adverbial-2 ((cat common) (distance near) (lex "way")))
   (end-adverbial-3 ((cat adv) (lex "often")))
   (end-adverbial-4 ((cat adv) (lex "now")))))


;; Tests: location as verbless clause
;;        time as finite clause
(def-test c6
  "As soon as you find it, keep it where accessible to all authorized users."
  ((cat clause)
   (mood imperative)
   (process ((type material) (lex "keep")))
   (partic ((agent ((index {^3 circum time partic processor index})))
            (affected ((cat personal-pronoun)))))
   (pred-modif
    ((location
      ((cat clause)
       (mood verbless)
       (controlled {^ partic carrier})
       (process ((type ascriptive)))
       (partic ((carrier ((index {^5 partic affected index})))
                (attribute ((cat ap)
                            (head ((lex "accessible")))
                            (qualifier ((cat pp)
                                        (prep ((lex "to")))
                                        (np ((cat common)
                                             (number plural)
                                             (definite no)
                                             (total +)
                                             (describer ((lex "authorized")))
                                             (head ((lex "user")))))))))))))))
   (circum ((time ((cat clause)
                   (mood bound-adverbial)
                   (binder ((lex "as soon as")))
                   (process ((type mental) (lex "find")))
                   (partic ((processor ((cat personal-pronoun) (person second)))
                            (phenomenon ((cat personal-pronoun)
                                         (index {^5 partic affected index}))))))))))


;; Tests: direction as PP
;;        distance as common
;;        time as present-participle clause
(def-test c7
  "After lifting weights, Tony biked up the mountain a few miles."
  ((cat clause)
   (tense past)
```

```
   (process ((type material) (effective no) (lex "bike")))
   (partic ((agent ((cat basic-proper) (lex "Tony")))))
   (pred-modif ((direction ((cat pp)
                            (prep ((lex "up")))
                            (np ((cat common) (lex "mountain")))))
                (distance ((cat common)
                           (lex "mile")
                           (definite no)
                           (orientation +)
                           (exact no)
                           (number plural)
                           (degree -)))))
   (circum ((time ((cat clause)
                   (mood present-participle)
                   (controlled {^ partic agent})
                   (binder ((lex "after")))
                   (process ((type material) (effective no) (lex "lift")))
                   (partic ((agent ((index {^5 partic agent index})))
                            (range ((cat common)
                                    (definite no)
                                    (number plural)
                                    (lex "weight")))))))))))


;; Tests: time as past-participle clause
;;        duration as adverb
(def-test c9
  "Once refused a new contract, Bo held out indefinitely."
  ((cat clause)
   (tense past)
   (process ((type composite) (relation-type locative) (lex "hold")))
   (partic ((agent ((cat basic-proper) (lex "Bo")))
            (located {^ agent})
            (location ((cat adv) (lex "out")))))
   (pred-modif ((duration ((cat adv) (lex "indefinitely")))))
   (circum ((time ((cat clause)
                   (mood past-participle)
                   (controlled {^ partic possessor})
                   (binder ((lex "once")))
                   (process ((type composite)
                             (relation-type possessive)
                             (lex "refuse")))
                   (partic ((possessor ((index {^5 partic processor index})))
                            (possessed ((cat common)
                                        (definite no)
                                        (describer ((lex "new")))
                                        (head ((lex "contract")))))
                            (affected {^ possessor})))))))))

(store-verbs '(("hold" "holds" "held" "holding" "held")))


;; Tests: duration as PP
;;        frequency as PP
```

```
(def-test c10
  "For two years, Bo worked out on Sundays."
  ((cat clause)
   (tense past)
   (process ((type material) (effective no) (lex "work out")))
   (partic ((agent ((cat basic-proper) (lex "Bo")))))
   (circum ((frequency ((cat pp)
                        (position end)
                        (prep ((lex "on")))
                        (np ((cat common)
                             (definite no)
                             (number plural)
                             (lex "Sunday")))))
            (duration ((cat pp)
                       (position front)
                       (np ((cat measure)
                            (quantity ((value 2)))
                            (unit ((lex "year")))))))))))


;; Tests: duration as finite clause
;;        reason as PP
(def-test c11
  "Because of his injury, Bo did not play until the playoffs started."
  ((cat clause)
   (tense past)
   (polarity negative)
   (process ((type material) (effective no) (lex "play")))
   (partic ((agent ((cat basic-proper) (lex "Bo") (gender masculine)))))
   (circum ((reason ((cat pp)
                     (np ((cat common)
                          (possessor ((cat personal-pronoun)
                                      (index {^5 partic agent index})))
                          (lex "injury")))))
            (duration ((cat clause)
                       (tense past)
                       (mood bound-adverbial)
                       (binder ((lex "until")))
                       (process ((type material)
                                 (agentive no)
                                 (effect-type creative)
                                 (lex "start")))
                       (partic ((created ((cat common)
                                          (number plural)
                                          (lex "playoff")))))))))))

;; Tests: duration as present-participle clause
;;        co-event as present-participle clause
;;        present-participle clause with subject
(def-test c12
  "With his knees hampering him, Blackman has not played since becoming a Knick."
  ((cat clause)
   (tense present-perfect)
   (polarity negative)
```

```
   (process ((type material) (effective no) (lex "play")))
   (partic ((agent ((cat basic-proper) (lex "Blackman") (gender masculine)))))
   (circum
    ((duration ((cat clause)
                (mood present-participle)
                (controlled {^ partic agent})
                (binder ((lex "since")))
                (process ((type composite) (relation-type ascriptive) (lex "become")))
                (partic ((agent ((index {^5 partic agent index})))
                         (carrier {^ agent})
                         (attribute ((cat common)
                                     (definite no)
                                     (head ((lex "Knick")))))))))
     (co-event ((cat clause)
                (mood present-participle)
                (binder ((lex "with")))
                (process ((type material) (lex "hamper")))
                (partic ((agent ((cat common)
                                 (possessor ((cat personal-pronoun)
                                             (index {^6 partic agent index})))
                                 (number plural)
                                 (lex "knee")))
                         (affected ((cat personal-pronoun)
                                    (index {^5 partic agent index}))))))))))


;; Tests: duration as verbless clause
;;        habitual co-event as finite clause with subject
(def-test c13
  "Whenever pain resurfaces, rest as long as necessary."
  ((cat clause)
   (mood imperative)
   (process ((type material) (effective no) (lex "rest")))
   (circum ((duration ((cat clause)
                       (mood verbless)
                       (binder ((lex "as long as")))
                       (process ((type ascriptive)))
                       (controlled {^ partic carrier})
                       (partic ((attribute ((cat ap) (lex "necessary")))))))
            (co-event ((cat clause)
                       (mood bound-adverbial)
                       (habitual yes)
                       (process ((type material) (effective no) (lex "resurface")))
                       (partic ((agent ((cat common)
                                        (countable no)
                                        (lex "pain")))))))))))


;; Tests: co-event as subjectless present-participle clause
;;        cat date
;;        cat address
;;        header adverbial
;;        inclusion as present-participle clause
(def-test c14
```

```
"Salt Lake City, Utah -- Karl Malone scored 28 points Saturday,
 including making 10 of 12 field goals, leading the Utah Jazz to a 105 - 95 win
 over the Los Angeles Clippers."
((cat clause)
 (tense past)
 (process ((type material) (effect-type creative) (lex "score")))
 (partic ((agent ((cat compound-proper)
                  (head ((cat person-name)
                         (first-name ((lex "Karl")))
                         (last-name ((lex "Malone")))))))
          (created ((cat measure)
                    (quantity ((value 28)))
                    (unit ((lex "point")))))))
 (circum
  ((time ((cat date) (day-name ((lex "Saturday")))))
   (location ((cat address)
              (position header)
              (punctuation ((after "--")))
              (city ((lex "Salt Lake City")))
              (state ((lex "Utah")))))
   (inclusion ((cat clause)
               (mood present-participle)
               (position end)
               (controlled {^ partic agent})
               (process ((type material) (effect-type creative) (lex "make")))
               (partic
                ((agent ((index {^5 partic agent index})))
                 (created ((cat partitive)
                           (part ((value 10) (digit yes)))
                           (part-of ((cat measure)
                                     (quantity ((value 12)))
                                     (unit ((lex "field goal")))))))))))
   (co-event
    ((cat clause)
     (mood present-participle)
     (position end)
     (controlled {^ partic agent})
     (process ((type composite)
               (relation-type locative)
               (lex "lead")))
     (partic
      ((agent ((index {^5 partic agent index})))
       (located ((cat compound-proper)
                 (number plural)
                 (head ((cat team-name)
                        (home ((lex "Utah")))
                        (franchise ((lex "Jazz")))))))
       (affected {^ located})
       (location
        ((cat pp)
         (prep ((lex "to")))
         (np ((cat common)
              (definite no)
              (classifier ((cat score)
```

```
                              (win ((value 105)))
                              (lose ((value 95)))))
                 (head ((lex "win")))
                 (qualifier
                  ((cat pp)
                   (prep ((lex "over")))
                   (np ((cat compound-proper)
                        (number plural)
                        (head ((cat team-name)
                               (home ((lex "Los Angeles")))
                               (franchise ((lex "Clipper"))))))))))))))))))))

(store-plurals '(("Jazz" "Jazz")))


;; Tests: co-event as subjectless past-participle clause
;;        opposition role
(def-test c15
  "Injured against the Giants, Bo missed 11 games."
  ((cat clause)
   (tense past)
   (process ((type material) (effective no) (lex "miss")))
   (partic ((agent ((cat basic-proper) (lex "Bo")))
            (range ((cat measure) (quantity ((value 11))) (unit ((lex "game")))))))
   (circum
    ((co-event
      ((cat clause)
       (mood past-participle)
       (controlled {^ partic affected})
       (binder none)
       (process ((type material) (lex "injure") (agentive no)))
       (partic ((affected ((index {^5 partic agent index})))))
       (circum ((opposition ((cat pp)
                             (np ((cat compound-proper)
                                  (number plural)
                                  (head ((cat team-name)
                                         (franchise ((lex "Giant"))))))))))))))))


;; Tests: co-event as verbless clause
;;        to-infinitive clause as adjective qualifier
;;        reason as adjunct finite clause
(def-test c16
  "Unable to play because he was injured, Bo stayed home."
  ((cat clause)
   (tense past)
   (process ((type composite) (relation-type locative) (lex "stay")))
   (partic ((agent ((cat basic-proper) (lex "Bo") (gender masculine)))
            (located {^ agent})
            (location ((cat adv) (lex "home")))))
   (circum
    ((co-event
      ((cat clause)
       (position front)
```

```
       (mood verbless)
       (binder none)
       (controlled {^ partic carrier})
       (process ((type ascriptive)))
       (partic
        ((carrier ((index {^5 partic agent index})))
         (attribute
          ((cat ap)
           (head ((lex "unable")))
           (qualifier ((cat clause)
                       (mood to-infinitive)
                       (controlled {^ partic agent})
                       (process ((type material) (effective no) (lex "play")))
                       (partic ((agent ((index {^5 carrier index})))))))))))
       (circum ((reason ((cat clause)
                         (mood bound-adverbial)
                         (tense past)
                         (process ((type ascriptive)))
                         (partic ((carrier ((cat personal-pronoun)
                                            (index {^7 partic located index})))
                                  (attribute ((cat verb)
                                              (ending past-participle)
                                              (lex "injure")))))))))))))))


;; Tests: reason as disjunct finite clause
;;        embedded reason
(def-test c17
  "Since he was unable to play because of injury, Bo stayed home."
  ((cat clause)
   (tense past)
   (process ((type composite) (relation-type locative) (lex "stay")))
   (partic ((agent ((cat basic-proper) (lex "Bo") (gender masculine)))
            (located {^ agent})
            (location ((cat adv) (lex "home")))))
   (circum
    ((reason
      ((cat clause)
       (position front)
       (mood bound-adverbial)
       (tense past)
       (binder ((lex "since")))
       (process ((type ascriptive)))
       (partic
        ((carrier ((cat personal-pronoun)
                   (index {^5 partic agent index})))
         (attribute
          ((cat ap)
           (head ((lex "unable")))
           (qualifier ((cat clause)
                       (mood to-infinitive)
                       (controlled {^ partic agent})
                       (process ((type material) (effective no) (lex "play")))
                       (partic ((agent ((index {^5 carrier index})))))
```

```
                      (circum ((reason ((cat pp)
                                        (np ((cat common)
                                             (denotation illness)
                                             (lex "injury"))))))))))))))))))))


;; Tests: result as finite clause
(def-test c18
  "Bo was unable to play because of injury, so he stayed home."
  ((cat clause)
   (tense past)
   (process ((type ascriptive)))
   (partic
    ((carrier ((cat basic-proper) (lex "Bo") (gender masculine)))
     (attribute ((cat ap)
                 (head ((lex "unable")))
                 (qualifier ((cat clause)
                             (mood to-infinitive)
                             (process ((type material) (effective no) (lex "play")))
                             (partic ((agent ((index {^5 carrier index})))))
                             (circum ((reason ((cat pp)
                                               (np ((cat common)
                                                    (denotation illness)
                                                    (lex "injury")))))))))))))
   (circum
    ((result ((cat clause)
              (mood bound-adverbial)
              (binder ((lex "so")))
              (tense past)
              (process ((type composite) (relation-type locative) (lex "stay")))
              (partic ((agent ((cat personal-pronoun)
                               (index {^5 partic carrier index})))
                       (located {^ agent})
                       (location ((cat adv) (lex "home")))))))))))


;; Tests: result as to-infinitive clause
;;        score as circumstance
(def-test c19
  "The Utah Jazz defeated the Los Angeles Clippers 105 - 95, to extend their winning
   streak to 11 games."
  ((cat clause)
   (tense past)
   (process ((type material) (lex "defeat")))
   (partic ((agent ((cat compound-proper)
                    (number plural)
                    (head ((cat team-name)
                           (home ((lex "Utah")))
                           (franchise ((lex "Jazz")))))))
            (affected ((cat compound-proper)
                       (number plural)
                       (head ((cat team-name)
                              (home ((lex "Los Angeles")))
                              (franchise ((lex "Clipper"))))))))))
```

```
   (pred-modif ((score ((cat score) (win ((value 105))) (lose ((value 95)))))))
   (circum ((result ((cat clause)
                     (mood to-infinitive)
                     (controlled {^ partic agent})
                     (process ((type composite)
                               (relation-type locative)
                               (lex "extend")))
                     (partic ((agent ((index {^5 partic agent index})))
                              (affected {^ located})
                              (located ((cat common)
                                        (possessor ((cat personal-pronoun)
                                                    (index {^3 agent index})))
                                        (describer ((cat verb)
                                                    (ending present-participle)
                                                    (lex "win")))
                                        (head ((lex "streak")))))
                              (location ((cat pp)
                                         (prep ((lex "to")))
                                         (np ((cat measure)
                                              (quantity ((value 11)))
                                              (unit ((lex "game"))))))))))))))


;; Tests: purpose as PP
(def-test c20
  "The Utah Jazz defeated the Los Angeles Clippers 105 - 95 for its 11th straight
   victory."
  ((cat clause)
   (tense past)
   (process ((type material) (lex "defeat")))
   (partic ((agent ((cat compound-proper)
                    (number plural)
                    (head ((cat team-name)
                           (home ((lex "Utah")))
                           (franchise ((lex "Jazz")))))))
            (affected ((cat compound-proper)
                       (number plural)
                       (head ((cat team-name)
                              (home ((lex "Los Angeles")))
                              (franchise ((lex "Clipper")))))))))
   (pred-modif ((score ((cat score) (win ((value 105))) (lose ((value 95)))))))
   (circum ((purpose ((cat pp)
                      (position end)
                      (np ((cat common)
                           (possessor ((cat personal-pronoun)
                                       (index {^4 partic agent index})))
                           (ordinal ((value 11)))
                           (classifier ((cat adj) (lex "straight")))
                           (head ((lex "victory"))))))))))


;; Tests: purpose as finite clause
;;        exception role
(def-test c21
```

```
  "Except Bo, all players do commercials so they can make extra money."
  ((cat clause)
   (process ((type material) (effective no) (lex "do")))
   (partic ((agent ((cat common)
                    (number plural)
                    (definite no)
                    (total +)
                    (lex "player")))
            (range ((cat common)
                    (number plural)
                    (definite no)
                    (lex "commercial")))))
   (circum ((exception ((cat pp) (np ((cat basic-proper) (lex "Bo")))))
            (purpose ((cat clause)
                      (mood bound-adverbial)
                      (binder ((lex "so")))
                      (epistemic-modality possible)
                      (process ((type material) (effect-type creative) (lex "make")))
                      (partic ((agent ((cat personal-pronoun)
                                       (index {^5 partic agent index})))
                               (created ((cat common)
                                         (countable no)
                                         (definite no)
                                         (classifier ((cat adj) (lex "extra")))
                                         (head ((lex "money")))))))))))))


;; Tests: behalf
;;        condition as finite clause
(def-test c23
  "If he is 100 percent fit, Bo will play for the Raiders on Sunday."
  ((cat clause)
   (tense future)
   (process ((type material) (effective no) (lex "play")))
   (partic ((agent ((cat basic-proper) (lex "Bo") (gender masculine)))))
   (circum ((behalf ((cat pp)
                     (np ((cat compound-proper)
                          (number plural)
                          (head ((cat team-name)
                                 (franchise ((lex "Raider")))))))))
            (time ((cat pp)
                   (position end)
                   (prep ((lex "on")))
                   (np ((cat basic-proper) (lex "Sunday")))))
            (condition ((cat clause)
                        (position front)
                        (mood bound-adverbial)
                        (process ((type ascriptive)))
                        (partic ((carrier ((cat personal-pronoun)
                                           (index {^5 partic agent index})))
                                 (attribute ((cat ap)
                                             (classifier ((cat measure)
                                                          (quantity ((value 100)))
                                                          (unit ((lex "percent")))))
```

```
                                             (head ((lex "fit"))))))))))))


;; Tests: condition as verbless clause
;;        substitution as PP
;;        concession as finite clause
(def-test c24
  "Even though he was not a starter at the beginning of the season, if fully fit,
   Bo will start tomorrow, instead of Smith."
  ((cat clause)
   (tense future)
   (process ((type material) (agentive no) (lex "start")))
   (partic ((affected ((cat basic-proper) (lex "Bo") (gender masculine)))))
   (circum
    ((concession ((cat clause)
                  (tense past)
                  (binder ((lex "even though")))
                  (polarity negative)
                  (process ((type ascriptive)))
                  (partic ((carrier ((cat personal-pronoun)
                                     (index {^5 partic affected index})))
                           (attribute ((cat common)
                                       (definite no)
                                       (lex "starter")))))
                  (circum ((time ((cat pp)
                                  (prep ((lex "at")))
                                  (np ((cat common)
                                       (head ((lex "beginning")))
                                       (qualifier ((cat pp)
                                                   (prep ((lex "of")))
                                                   (np ((cat common)
                                                        (lex "season")))))))))))))
     (condition ((cat clause)
                 (position front)
                 (mood verbless)
                 (process ((type ascriptive)))
                 (controlled {^ partic carrier})
                 (partic ((carrier ((index {^5 partic affected index})))
                          (attribute ((cat ap)
                                      (modifier ((cat adv) (lex "fully")))
                                      (head ((lex "fit")))))))))
     (substitution ((cat pp) (np ((cat basic-proper) (lex "Smith")))))
     (time ((cat adv) (lex "tomorrow")))))))


;; Tests: condition as past-participle clause
;;        concession as PP
(def-test c25
  "If defeated tonight, the Raiders will not make the playoffs, despite their better
   division record."
  ((cat clause)
   (tense future)
   (polarity negative)
```

```
   (process ((type material) (effective no) (lex "make")))
   (partic ((agent ((cat compound-proper)
                    (number plural)
                    (head ((cat team-name)
                           (franchise ((lex "Raider")))))))
            (range ((cat common) (number plural) (lex "playoff")))))
   (circum ((condition ((cat clause)
                        (mood past-participle)
                        (controlled {^ partic affected})
                        (process ((type material) (lex "defeat")))
                        (partic ((affected ((index {^5 partic agent index})))))
                        (circum ((time ((cat adv) (lex "tonight")))))))
            (concession ((cat pp)
                         (prep ((lex "despite")))
                         (np ((cat common)
                              (possessor ((cat personal-pronoun)
                                          (index {^5 partic agent index})))
                              (describer ((lex "better")))
                              (classifier ((lex "division")))
                              (head ((lex "record")))))))))))


;; Tests: time as verbless clause
;;        matter role
;;        manner as adverb
;;        distance as measure
;;        destination as finite clause
;;        substitution as present-participle clause
(def-test c26
  "Instead of panicking, when in doubt concerning his position, Bo cautiously
   retreated five miles towards where he came from."
  ((cat clause)
   (tense past)
   (process ((type material) (effective no) (lex "retreat")))
   (partic ((agent ((cat basic-proper) (lex "Bo") (gender masculine)))))
   (pred-modif ((distance ((cat measure)
                           (quantity ((value 5)))
                           (unit ((lex "mile")))))
                (destination ((cat clause)
                              (mood bound-adverbial)
                              (tense past)
                              (binder ((lex "towards where")))
                              (process ((type material)
                                        (effective no)
                                        (lex "come from")))
                              (partic ((agent ((cat personal-pronoun)
                                               (index {^5 partic agent index})))))))))
   (circum
    ((time ((cat clause)
            (mood verbless)
            (controlled {^ partic carrier})
            (process ((type ascriptive)))
            (partic ((carrier ((index {^5 partic agent index})))
                     (attribute ((cat pp)
```

```
                                (prep ((lex "in")))
                                (np ((cat common)
                                     (denotation zero-article-thing)
                                     (lex "doubt")))))))
           (circum ((matter ((cat pp)
                             (prep ((lex "concerning")))
                             (np ((cat common)
                                  (possessor ((cat personal-pronoun)
                                              (index {^5 partic carrier index})))
                                  (head ((lex "position")))))))))))
    (manner ((cat adv) (lex "carefully")))
    (substitution ((cat clause)
                   (mood present-participle)
                   (controlled {^ partic processor})
                   (process ((type mental) (transitive no) (lex "panic")))
                   (partic ((processor ((index {^5 partic agent index})))))))))))

(store-verbs '(("come from" "comes from" "came from" "coming from" "came from")))
(store-verbs '(("panic" "panics" "panicked" "panicking" "panicked")))


;; Tests: duration as past-participle clause
;;        habitual co-event as verbless clause
;;        manner as PP
;;        means as PP
(def-test c27
  "Whenever in doubt, resist with obstination until coerced by force."
  ((cat clause)
   (mood imperative)
   (process ((type material) (effective no) (lex "resist")))
   (partic ((agent ((cat personal-pronoun) (person second)))))
   (pred-modif ((manner ((cat pp)
                         (np ((cat common) (lex "obstination") (countable no)))))))
   (circum
    ((duration ((cat clause)
                (position end)
                (mood past-participle)
                (binder ((lex "until")))
                (process ((type material) (lex "coerce")))
                (controlled {^ partic affected})
                (partic ((affected ((index {^5 partic agent index})))))
                (pred-modif ((means ((cat pp)
                                     (np ((cat common) (lex "force")))))))))
     (co-event ((cat clause)
                (position front)
                (mood verbless)
                (habitual yes)
                (process ((type ascriptive)))
                (controlled {^ partic carrier})
                (partic ((carrier ((index {^5 partic agent index})))
                         (attribute ((cat pp)
                                     (prep ((lex "in")))
                                     (np ((cat common)
                                          (denotation zero-article-thing)
```

```
                                    (lex "doubt"))))))))))))


;; Tests: negative comparison as PP
;;        concessive-condition as finite clause
(def-test c28
  "Unlike Barkley, Jordan can dominate, even if he is not 100 percent fit."
  ((cat clause)
   (epistemic-modality possible)
   (process ((type material) (effective no) (lex "dominate")))
   (partic ((agent ((cat basic-proper) (lex "Jordan") (gender masculine)))))
   (circum ((comparison ((cat pp)
                         (position front)
                         (comp-polarity -)
                         (np ((cat basic-proper)
                              (lex "Barkley")))))
            (concessive-condition
             ((cat clause)
              (mood bound-adverbial)
              (polarity negative)
              (process ((type ascriptive)))
              (partic ((carrier ((cat personal-pronoun)
                                 (index {^5 partic agent index})))
                       (attribute ((cat ap)
                                   (classifier ((cat measure)
                                                (quantity ((value 100)))
                                                (unit ((lex "percent")))))
                                   (head ((lex "fit"))))))))))))


;; Tests: concessive-condition as present-participle clause
;;        positive instrument
;;        contrast as PP
(def-test c29
  "As opposed to pure penetrators, Jordan can kill you with his three point shot,
   even if suffering from tendinitis."
  ((cat clause)
   (epistemic-modality possible)
   (process ((type material) (lex "kill")))
   (partic ((agent ((cat basic-proper) (lex "Jordan") (gender masculine)))
            (affected ((cat personal-pronoun) (person second)))))
   (pred-modif ((instrument ((cat pp)
                             (np ((cat common)
                                  (possessor ((cat personal-pronoun)
                                              (index {^5 partic agent index})))
                                  (classifier ((cat measure)
                                               (quantity ((value 3)))
                                               (unit ((lex "point")))))
                                  (head ((lex "shot")))))))))
   (circum ((contrast ((cat pp)
                       (prep ((lex "as opposed to")))
                       (np ((cat common)
                            (number plural)
                            (definite no)
```

```
                              (describer ((lex "pure")))
                              (head ((lex "penetrator")))))))
             (concessive-condition
              ((cat clause)
               (mood present-participle)
               (controlled {^ partic processor})
               (process ((type mental) (lex "suffer from")))
               (partic ((processor ((index {^5 partic agent index})))
                        (phenomenon ((cat common)
                                     (denotation illness)
                                     (lex "tendinitis")))))))))))

(store-verbs '(("suffer from" "suffers from" "suffered from"
                "suffering from" "suffered from")))



;; Tests: concessive-condition as verbless clause
;;        contrast as finite clause
;;        present-participle clause as adjective qualifier
(def-test c30
  "Even if capable of playing forward, Magic was a guard, whereas Bird was a true
   forward."
  ((cat clause)
   (tense past)
   (process ((type ascriptive)))
   (partic ((carrier ((cat basic-proper) (lex "Magic")))
            (attribute ((cat common) (definite no) (lex "guard")))))
   (circum ((contrast ((cat clause)
                       (tense past)
                       (mood bound-adverbial)
                       (process ((type ascriptive)))
                       (partic ((carrier ((cat basic-proper) (lex "Bird")))
                                (attribute ((cat common)
                                            (definite no)
                                            (classifier ((lex "true")))
                                            (head ((lex "forward")))))))))
            (concessive-condition
             ((cat clause)
              (mood verbless)
              (process ((type ascriptive)))
              (controlled {^ partic carrier})
              (partic
               ((carrier ((index {^5 partic carrier index})))
                (attribute
                 ((cat ap)
                  (head ((lex "capable")))
                  (qualifier
                   ((cat clause)
                    (mood present-participle)
                    (binder ((lex "of")))
                    (controlled {^ partic agent})
                    (process ((type material) (effective no) (lex "play")))
                    (partic ((agent ((index {^5 partic carrier index})))
```

```
                              (range ((cat common)
                                      (denotation zero-article-thing)
                                      (lex "forward")))))))))))))))))


;; Tests: concessive-condition as past-participle clause
;;        positive comparison as PP
;;        past-participle clause with agent
(def-test c31
  "Like Jordan, Drexler can kill you from outside, even if slowed down by injury."
  ((cat clause)
   (epistemic-modality possible)
   (process ((type material) (lex "kill")))
   (partic ((agent ((cat basic-proper) (lex "Drexler")))
            (affected ((cat personal-pronoun) (person second)))))
   (circum ((comparison ((cat pp) (np ((cat basic-proper) (lex "Jordan")))))
            (concessive-condition
             ((cat clause)
              (mood past-participle)
              (controlled {^ partic affected})
              (process ((type material)
                        (lex "slow down")
                        (agentless no)))
              (partic ((agent ((cat common) (denotation illness) (lex "injury")))
                       (affected ((index {^5 partic agent index})))))))
            (location ((cat pp)
                       (prep ((lex "from")))
                       (np ((cat common)
                            (denotation zero-article-thing)
                            (lex "outside")))))))))

(store-verbs '(("slow down" "slows down" "slowed down" "slowing down" "slowed down")))


;; Tests: concession as past-participle clause
;;        positive accompaniment
(def-test c32
  "Although not wanted by the Knicks, Kimble was traded with Smith for Jackson."
  ((cat clause)
   (tense past)
   (process ((type material) (lex "trade") (voice passive)))
   (partic ((agent ((cat compound-proper)
                    (number plural)
                    (head ((cat team-name) (franchise ((lex "Knick")))))))
            (affected ((cat basic-proper) (lex "Kimble")))))
   (circum ((accompaniment ((cat pp)
                            (np ((cat basic-proper) (lex "Smith")))))
            (concession
             ((cat clause)
              (position front)
              (mood past-participle)
              (polarity negative)
              (process ((type mental) (lex "want") (agentless no)))
              (controlled {^ partic phenomenon})
```

```
              (partic ((processor {^4 partic agent})
                       (phenomenon ((index {^5 partic affected index})))))))
            (purpose ((cat pp)
                      (position end)
                      (np ((cat basic-proper) (lex "Jackson")))))))))


;; Tests: concession as present-participle clause
;;        negative accompaniment
;;        comparison as finite clause
(def-test c33
  "Although playing without Bo, the Raiders won as they did with him."
  ((cat clause)
   (tense past)
   (process ((type material) (effective no) (lex "win")))
   (partic ((agent ((cat compound-proper)
                    (number plural)
                    (head ((cat team-name)
                           (franchise ((lex "Raider")))))))))
   (pred-modif
    ((comparison
      ((cat clause)
       (tense past)
       (mood bound-adverbial)
       (binder ((lex "as")))
       (process ((type material) (effective no) (lex "do")))
       (partic ((agent ((cat personal-pronoun)
                        (index {^5 partic agent index})))))
       (circum
        ((accompaniment
          ((cat pp)
           (np ((cat personal-pronoun)
                (index {^6 circum concession circum accompaniment np index})))))))))))
   (circum ((concession ((cat clause)
                         (mood present-participle)
                         (controlled {^ partic agent})
                         (process ((type material) (effective no) (lex "play")))
                         (partic ((agent ((index {^5 partic agent index})))))
                         (circum ((accompaniment ((cat pp)
                                                  (accomp-polarity -)
                                                  (np ((cat basic-proper)
                                                       (gender masculine)
                                                       (lex "Bo"))))))))))))

(store-verbs '(("win" "wins" "won" "winning" "won")))

;; Tests: concession as verbless clause
;;        addition as PP
(def-test c34
  "Although in need of a center, they drafted a guard in addition to Flint."
  ((cat clause)
   (tense past)
   (process ((type composite) (relation-type possessive) (lex "draft")))
   (partic ((agent ((cat personal-pronoun) (number plural)))
```

```
           (possessor {^ agent})
           (possessed ((cat common) (definite no) (lex "guard")))))
   (circum ((addition ((cat pp)
                       (prep ((lex "in addition to")))
                       (np ((cat basic-proper) (lex "Flint")))))
           (concession ((cat clause)
                        (mood verbless)
                        (process ((type ascriptive)))
                        (controlled {^ partic carrier})
                        (partic ((carrier ((index {^5 partic agent index})))
                                 (attribute ((cat pp)
                                             (prep ((lex "in need of")))
                                             (np ((cat common)
                                                  (definite no)
                                                  (lex "center"))))))))))))


;; Tests: negative instrument
;;        means as present-participle clause and PP
;;        standard and perspective roles
;;        addition as present-participle clause
(def-test c35
  "They filled all their holes without a draft pick, by acquiring Bowman, whose passing
   game is exceptional for a seven footer, through free-agency in addition to trading
   for Ashbliss, who remains incomparable as a defensive stopper."
  ((cat clause)
   (tense past)
   (process ((type material) (lex "fill")))
   (partic ((agent ((cat personal-pronoun) (number plural)))
           (affected ((cat common)
                      (number plural)
                      (possessor ((cat personal-pronoun)
                                  (index {^3 agent index})))
                      (head ((lex "hole")))))))
   (pred-modif ((instrument ((cat pp)
                             (instr-polarity -)
                             (np ((cat common)
                                  (definite no)
                                  (classifier ((lex "draft")))
                                  (head ((lex "pick")))))))))
   (circum
    ((means
      ((cat clause)
       (position end)
       (mood present-participle)
       (process ((type composite) (relation-type possessive) (lex "acquire")))
       (controlled {^ partic agent})
       (partic
        ((agent ((index {^5 partic agent index})))
         (possessor {^ agent})
         (possessed ((cat compound-proper)
                     (head ((cat person-name)
                            (last-name ((lex "Bowman")))))
                     (qualifier
```

```
                        ((cat clause)
                         (mood relative)
                         (restrictive no)
                         (process ((type ascriptive)))
                         (scope {^ partic carrier possessor})
                         (partic ((carrier ((cat common)
                                            (classifier ((cat verb)
                                                         (ending present-participle)
                                                         (lex "pass")))
                                            (head ((lex "game")))))
                                  (attribute ((cat ap) (lex "exceptional")))))
                         (circum ((standard ((cat pp)
                                             (np ((cat common)
                                                  (definite no)
                                                  (lex "seven footer"))))))))))))
       (pred-modif ((means ((cat pp)
                            (prep ((lex "through")))
                            (np ((cat common) (lex "free agency")))))))
       (circum
        ((addition
          ((cat clause)
           (position end)
           (mood present-participle)
           (controlled {^ partic agent})
           (process ((type composite) (relation-type possessive) (lex "trade for")))
           (partic
            ((agent ((index {^5 partic agent index})))
             (possessor {^ agent})
             (possessed
              ((cat compound-proper)
               (head ((cat person-name)
                      (last-name ((lex "Ashbliss")))))
               (qualifier
                ((cat clause)
                 (mood relative)
                 (process ((type ascriptive) (lex "remain")))
                 (scope {^ partic carrier})
                 (partic ((attribute ((cat ap) (lex "incomparable")))))
                 (circum
                  ((perspective
                    ((cat pp)
                     (np ((cat common)
                          (definite no)
                          (classifier ((lex "defensive")))
                          (head ((lex "stopper")))))))))))))))))))))))

(store-verbs '(("trade for" "trades for" "traded for" "trading for" "traded for")))

;; Tests: comparison as verbless clause
;;        means as adverb
(def-test c36
  "They treated him homeopathically as if afraid of conventional treatments."
  ((cat clause)
   (tense past)
```

```
   (process ((type material) (lex "treat")))
   (partic ((agent ((cat personal-pronoun) (number plural)))
            (affected ((cat personal-pronoun) (gender masculine)))))
   (pred-modif
    ((means ((cat adv) (lex "homeopathically")))
     (comparison
      ((cat clause)
       (mood verbless)
       (process ((type ascriptive)))
       (controlled {^ partic carrier})
       (partic ((carrier ((index {^5 partic agent index})))
                (attribute ((cat ap)
                            (head ((lex "afraid")))
                            (qualifier ((cat pp)
                                        (prep ((lex "of")))
                                        (np ((cat common)
                                             (number plural)
                                             (definite no)
                                             (describer ((lex "conventional")))
                                             (head ((lex "treatment")))))))))))))))))


;; Tests: comparison as past-participle clause
(def-test c37
  "They treated him homeopathically as if frightened by conventional treatments."
  ((cat clause)
   (tense past)
   (process ((type material) (lex "treat")))
   (partic ((agent ((cat personal-pronoun) (number plural)))
            (affected ((cat personal-pronoun) (gender masculine)))))
   (pred-modif
    ((means ((cat adv) (lex "homeopathically")))
     (comparison
      ((cat clause)
       (mood past-participle)
       (process ((type mental) (lex "frighten") (agentless no)))
       (controlled {^ partic phenomenon})
       (partic ((phenomenon ((index {^5 partic agent index})))
                (processor ((cat common)
                            (number plural)
                            (definite no)
                            (describer ((lex "conventional")))
                            (head ((lex "treatment")))))))))))))

(store-verbs '(("frighten" "frightens" "frightened" "frightening" "frightened")))


;; Tests: comparison as present-participle clause
(def-test c38
  "They treated him homeopatically as if distrusting conventional treatments."
  ((cat clause)
   (tense past)
   (process ((type material) (lex "treat")))
   (partic ((agent ((cat personal-pronoun) (number plural)))
```

```
          (affected ((cat personal-pronoun) (gender masculine)))))
 (pred-modif
  ((means ((cat adv) (lex "homeopatically")))
   (comparison
    ((cat clause)
     (mood present-participle)
     (process ((type mental) (lex "distrust")))
     (controlled {^ partic processor})
     (partic ((processor ((index {^5 partic agent index})))
              (phenomenon ((cat common)
                           (number plural)
                           (definite no)
                           (describer ((lex "conventional")))
                           (head ((lex "treatment")))))))))))))


;; Tests: comparison as to-infinitive clause
;;        past-participle clause with subject
(def-test c39
  "The analysis completed, they treated him homeopatically as if to avoid conventional
   treatments."
  ((cat clause)
   (tense past)
   (process ((type material) (lex "treat")))
   (partic ((agent ((cat personal-pronoun) (number plural)))
            (affected ((cat personal-pronoun) (gender masculine)))))
   (pred-modif
    ((means ((cat adv) (lex "homeopatically")))
     (comparison
      ((cat clause)
       (mood to-infinitive)
       (process ((type material) (effective no) (lex "avoid")))
       (controlled {^ partic agent})
       (partic ((agent ((index {^5 partic agent index})))
                (range ((cat common)
                        (number plural)
                        (definite no)
                        (describer ((lex "conventional")))
                        (head ((lex "treatment")))))))))))
   (circum ((co-event ((cat clause)
                       (mood past-participle)
                       (process ((type material) (lex "complete")))
                       (partic ((affected ((cat common) (lex "analysis")))))))))))


;; Alternative analysis, tests: verbless clause with subject
(def-test c39bis
  "The analysis completed, they treated him homeopatically as if to avoid conventional
   treatments."
  ((cat clause)
   (tense past)
   (process ((type material) (lex "treat")))
   (partic ((agent ((cat personal-pronoun) (number plural)))
            (affected ((cat personal-pronoun) (gender masculine)))))
```

```
   (pred-modif
    ((means ((cat adv) (lex "homeopatically")))
     (comparison
      ((cat clause)
       (mood to-infinitive)
       (process ((type material) (effective no) (lex "avoid")))
       (partic ((agent ((index {^5 partic agent index})))
                (range ((cat common)
                        (number plural)
                        (definite no)
                        (describer ((lex "conventional")))
                        (head ((lex "treatment")))))))))))
   (circum ((co-event ((cat clause)
                       (mood verbless)
                       (process ((type ascriptive)))
                       (partic ((carrier ((cat common) (lex "analysis")))
                                (attribute ((cat verb)
                                            (ending past-participle)
                                            (lex "complete")))))))))))


;; Tests: present-participle clause as noun qualifier
;;        past-participle clause as noun qualifier
;;        to-infinitive clause as noun qualifier
(def-test c40
  "The first team winning two games will be the team to take the title recognized by
   the WSL."
((cat clause)
 (tense future)
 (process ((type ascriptive) (relation-mode equative)))
 (partic
  ((identified ((cat common)
                (ordinal ((value 1)))
                (head ((lex "team")))
                (qualifier ((cat clause)
                            (mood present-participle)
                            (process ((type material) (effective no) (lex "win")))
                            (controlled {^ partic agent})
                            (partic ((agent ((index {^5 identified index})))
                                     (range ((cat measure)
                                             (quantity ((value 2)))
                                             (unit ((lex "game")))))))))))
   (identifier
    ((cat common)
     (head ((lex "team")))
     (qualifier
      ((cat clause)
       (mood to-infinitive)
       (process ((type material) (effective no) (lex "take")))
       (controlled {^ partic agent})
       (partic
        ((agent ((index {^5 identifier index})))
         (range ((cat common)
                 (head ((lex "title")))
```

```
                   (qualifier
                    ((cat clause)
                     (mood past-participle)
                     (process ((type mental) (lex "recognize") (agentless no)))
                     (controlled {^ partic phenomenon})
                     (partic ((processor ((cat basic-proper) (lex "the WSL")))
                              (phenomenon ((index {^5 range index}))))))))))))))))))))


;; Tests: for-to-infinitive clause with subject as adverbial
(def-test c41
  "I did it for you to graduate."
  ((cat clause)
   (tense past)
   (process ((type material) (effect-type creative) (lex "do")))
   (partic ((agent ((cat personal-pronoun) (person first)))
            (created ((cat personal-pronoun)))))
   (circum
    ((purpose ((cat clause)
               (mood for-to-infinitive)
               (binder none)
               (process ((type mental) (transitive no) (lex "graduate")))
               (partic ((processor ((cat personal-pronoun) (person second))))))))))


;; Tests: backward compatibility of condition w/ cond-relater
(def-test c42
  "If the compartment is dirty, then clean it."
  ((cat clause)
   (mood imperative)
   (proc ((type material) (lex "clean")))
   (partic ((affected ((cat personal-pronoun)))))
   (relaters ((cond ((lex "then")))))
   (circum ((condition ((cat clause)
                        (position front)
                        (then yes)
                        (proc ((type ascriptive)))
                        (partic ((carrier ((cat common) (lex "compartment")))
                                 (attribute ((cat ap) (lex "dirty")))))))))))

;; Tests: backward compatibility of from-loc
(def-test c43
  "It comes from the KY57."
  ((cat clause)
   (proc ((type material) (lex "come")))
   (partic ((agent ((cat personal-pronoun)))))
   (circum ((from-loc ((cat common) (lex "KY57")))))))

(def-test c44
  "It comes from the KY57."
  ((cat clause)
   (proc ((type material) (lex "come")))
   (partic ((agent ((cat personal-pronoun)))))
   (pred-modif ((origin ((cat pp)
```

```
(np ((cat common) (lex "KY57")))))))))
```

Appendix C

Example runs of STREAK

In this appendix I present example runs of the generation system STREAK. Two example runs were already presented in Section 4.4. I start by presenting again these two runs more in depth. I provide the FDs encoding the draft presentation and/or the semantic input for those runs and I discuss how they influence the application of revision rules. I then present three additional runs illustrating other interesting aspects of the system, namely its paraphrasing power (both at draft-time and revision-time) and the application of the same revision rules in different contexts.

C.1 Original example runs with input and draft representation

C.1.1 Example run 1: Chaining different revision rules to generate a complex sentence

Run 1 shown in Fig. C.1 illustrates three aspects of STREAK:

- How it generates a complex lead sentence, by first producing a simple sentence containing only the obligatory fixed facts and then incrementally incorporating the complementary floating facts through a series of revisions.
- How it controls the revision process and decides when to halt it.
- The variety of revision rules it implements, since a different one is used at each generation increment.

This run was already commented in Section 4.4.1. In this section, I present this run at further depth by providing:

- The input DSS to the drafting pass.
- The input additional DSS to each revision increment.
- The internal three-layer representation of the final draft (after the revision process completed).

C.1.1.1 Building an initial draft

In Fig. C.1, the first line contains calls to the top-level functions for the draft and revision stages. As input, the function `draft` takes three arguments:

- The DSS FD containing the input fixed facts to convey in the draft. In the example of Fig. C.1 this input DSS is built by calling the function `dssF0`. The code of this function is given in Fig. C.3.

```
> (revise (draft (dssF0) :sss (form-flag1)) (float-stack-F) :verbose T)

Draft 0:
Dallas, TX -- Charles Barkley registered 42 points Friday night as the Phoenix Suns
routed the Dallas Mavericks 123 - 97.

Draft 1 (lex-num = 27  depth = 7):
Dallas, TX -- Charles Barkley registered 42 points Friday night as the Phoenix Suns
handed the Dallas Mavericks their 27th defeat in a row at home 123 - 97.

Draft 2 (lex-num = 29  depth = 8):
Dallas, TX -- Charles Barkley registered 42 points Friday night as the Phoenix Suns
handed the Dallas Mavericks their franchise worst 27th defeat in a  row at home
123 - 97.

Draft 3 (lex-num = 34  depth = 8):
Dallas, TX -- Charles Barkley registered 42 points and Danny Ainge added 21 Friday
night as the Phoenix Suns handed the Dallas Mavericks their franchise worst 27th defeat
in a row at home 123 - 97.

Draft 4 (lex-num = 39  depth = 8):
Dallas, TX -- Charles Barkley registered 42 points and Danny Ainge came off the bench
to add 21 Friday night as the Phoenix Suns handed the Dallas Mavericks their franchise
worst 27th defeat in a row at home 123 - 97.

Draft 5 (lex-num = 43  depth = 8):
Dallas, TX -- Charles Barkley matched his season record with 42 points and Danny Ainge
came off the bench to add 21 Friday night as the Phoenix Suns handed the Dallas
Mavericks their franchise worst 27th defeat in a row at home 123 - 97.

Draft 6 (lex-num = 46  depth = 8):

((SSS ((DSS .... ))))
>
```

Figure C.1: Applying a different rule at each revision increment

- A partial SSS constraining the form of the output draft by specifying some desired features in the initial draft plan. The phrase planner can only choose options that are compatible with this partial specification of its output. This argument is optional and introduced by the keyword `:sss`. These surface form constraints are generated by calls to functions such as `form-flag1` shown in Fig. C.2.
- A partial DGS constraining the form of the output draft by specifying some desired features in the initial draft skeletal lexico-syntactic tree. The lexicalizer can only choose options that are compatible with this partial specification of its output. This argument is optional and introduced by the keyword `:dgs`[1].

`draft` returns a three-layer FD encoding the initial draft and as a side effect prints the natural language sentence resulting from the unification of this FD with SURGE.

The function `revise` takes three arguments:

- A three-layer FD encoding the initial draft.
- A list of lists of floating facts to attempt to incorporate to the draft. Each sublist contains a set of related floating facts. These sublists are ordered by decreasing importance of the facts they contain. Floating fact lists of lists are generated by calls to functions such as `float-stack-F` shown at the top of Fig. C.4. The facts contained in this list of list are shown in figures C.4 to C.8.
- The `:verbose` flag, which, when set to T, prints after each revision increment the lexical length (lex-num) and syntactic depth (depth) of the revised draft.

As explained in Section 2.4 an initial draft contains four fixed facts: the main statistic of a player from the winning team, the result of the game (including its final score), its location and its date. Following the corpus observations, STREAK always conveys the location as a header. The sentence itself thus contains three main constituents, one for each of the three remaining fixed facts. The form flag specifies the type of rhetorical relations that the phrase planner can use to group these three constituents. `form-flag1` specifies that:

- The draft must be hypotactically structured (indicated by the presence of a `rels` feature at the top-level).
- Two dependent constituents must be grouped in an paratactic structure itself linked to the main constituent by a temporal relation (indicated by the presence of a feature `(time ((elt ...)))` under the `rels` feature).
- The second element in this paratactic structure must be *itself hypotactically structured* (indicated by the embedded `(struct hypotax)` feature).

When unified with the possible top-level draft structures observed in the corpus and encoded in the phrase planner of STREAK, this specification results in sentences like *Draft 0* at the top of Fig. C.1. In such sentences, the main clause conveys the main statistic with a list of two dependent constituents as a temporal adjunct. The first element of this list is the nominal conveying the date of the game and the second is the clause conveying the game result.

`form-flag2` specifies that:

- The draft must be hypotactically structured (indicated by the presence of a `rels` feature at the top-level).
- Two dependent constituents must be grouped in an paratactic structure itself linked to the main constituent by a temporal relation (indicated by the presence of a feature `(time ((elt ...)))` under the `rels` feature).
- The second element in this paratactically structure must be *structured as an event* (indicated by the embedded `((struct event))` feature).

[1] The two optional arguments `:sss` and `:dgs` provide a basic facility for systematically testing the paraphrasing power of STREAK. The development of more powerful facilities has been left for future work and is discussed in Section 7.2.2.2

```
;; Surface form constraint forcing the drafter to:
;; - attach the game result to the main game statistic as a time adverbial
;; - realize the game result by a full verb clause
;; e.g., ``as WINNER defeated LOSER SCORE''
(defun form-flag1 () (setf form-flag1 '((rels ((co-occur none)
                                               (time ((elts ((cdr ((car ((struct hypotax)))))))))))))

;; Surface form constraint forcing the drafter to:
;; - attach the game result to the main game statistic as a time adverbial
;; - realize the game result by a support verb clause
;; e.g., ``while WINNER claimed a SCORE win over LOSER''
(defun form-flag2 () (setf form-flag2 '((rels ((co-occur none)
                                               (time ((elts ((cdr ((car ((struct event)))))))))))))

;; Surface form constraint forcing the drafter to:
;; - attach the game result as a co-event subordinate clause
;; e.g., ``leading WINNER to a SCORE win over LOSER''
(defun form-flag3 () (setf form-flag3 '((rels ((time ((elts none))))))))
```

Figure C.2: Form flags used in the example runs

It thus shares the two first constraints with `form-flag2` and differs in the third. In contrast, `form-flag3` specifies the converse of the first two constraints of `form-flag1` and `form-flag2`, *i.e.,* that the draft must be either paratactically structured or, if hypotactically structured, then the dependent constituent must not be linked to the head by a temporal relation (as indicated by the `none` meta-value under the path `{rels time elts}`).

C.1.1.2 Revising the initial draft

The first sub-element in `float-stack-F` is an historical background fact of type streak extension. It notes that the reported game marks the 27th time that Dallas is defeated on its home turf. To add this fact to *Draft 0*, STREAK uses the non-monotonic `Nominalization` revision rule whose code was given and explained in detail in Section 4.3.3.1. This rule is applied to the initial draft game result clause "*the Phoenix Suns routed the Dallas Mavericks*". It replaces the full verb clause pattern "WINNER rout LOSER" conveying the game result in *Draft 0*, by the semantically equivalent support verb clause pattern "WINNER hand LOSER a defeat" in *Draft 1*. Since the game result is now realized by an NP, the expression of its consequence, the updated length of Dallas' losing streak, can be concisely conveyed by adjoining the discontinuous ordinal "*27th ... in a row*" to the NP head "*defeat*". The restriction of this streak to home games is conveyed by adjoining another modifier, the PP qualifier "*at home*". The type of nominalization rule used for this revision is thus: `Nominalization with Ordinal and Qualifier Adjoin`.

The second sub-element in `float-stack-F` is another historical background fact, of type record breaking. It brings additional information about the preceding streak extension fact, by noting that as a result of this latest extension this streak is now of record length. To add this fact to *Draft 1*, STREAK uses the monotonic revision rule `Adjoin of Classifier to Nominal`. This rule is applied on the very nominal that was created during the preceding nominalization revision ("*their 27th defeat in a row at home*") modifying it with the classifier "*franchise worst*" that expresses its record breaking nature. This illustrates how the choice of a revision rule for a given increment constrains in some cases the range of choices for subsequent increments. This type of `Adjoin` revision rule allows for a very concise expression of the added floating fact. It does not change the syntactic depth of the draft and lengthens it by only two words. After this addition, the draft is only 29 words long, still comfortably below the 45 word limit observed in the corpus.

The third sub-element in `float-stack-F` is a non-historical fact of type additional statistic. To add this

```
(defun dssF0 ()
 ;; Dallas, TX -- Charles Barkley scored 42 points Friday night as the Phoenix
 ;; Suns defeated the Dallas Mavericks 123 - 97.
 (setf dssF0
  `((deepsemcat entity)
    (concept game) (token pho-at-dal)
    (attrs ((setting ((ents ((addr ((deepsemcat entity)
                                    (concept address) (token dallas-tx)
                                    (attrs ((city "Dallas") (state "TX")))))
                             (date ((deepsemcat entity)
                                    (concept date) (token fri-nite)
                                    (attrs ((day-name "Friday") (day-part night)))))))))
            (stats ((ents ((stat0-ca ((deepsemcat entity)
                                       (concept player) (token barkley)
                                       (attrs ((first-name "Charles")
                                               (last-name "Barkley")))))
                            (stat0 ((deepsemcat entity)
                                    (concept game-stat) (token barkley-pt-at-dal)
                                    (attrs ((value 42) (unit pt)))))))
                    (rels ((stat0 ((deepsemcat relation)
                                   (role game-stat-rel) (token barkley-scoring-at-dal)
                                   (args ((carrier {^4 ents stat0-ca})
                                          (stat {^4 ents stat0})))))))))
            (results ((ents ((host ((deepsemcat entity)
                                     (concept team) (token mavs)
                                     (attrs ((home "Dallas") (franchise "Maverick")))))
                              (visitor ((deepsemcat entity)
                                        (concept team) (token suns)
                                        (attrs ((home "Phoenix") (franchise "Sun")))))
                              (score ((deepsemcat entity)
                                      (concept score) (token pho-at-dal-score)
                                      (attrs ((win 123) (lose 97)))))))
                      (rels ((winner ((deepsemcat relation)
                                      (role winner) (token pho-at-dal-winner)
                                      (args ((game top-level)
                                             (winner {^4 ents visitor})))))
                              (loser ((deepsemcat relation)
                                      (role loser) (token pho-at-dal-loser)
                                      (args ((game top-level)
                                             (loser {^4 ents host})))))
                              (result ((deepsemcat relation)
                                       (role beat) (token pho-at-dal-result)
                                       (args ((winner {^4 ents visitor})
                                              (loser {^4 ents host}))))))))))))))
```

Figure C.3: Input containing the four fixed facts for Draft 0 in *Run 1*

```
(defun float-stack-F () (setf float-stack-F `((,adssF1 ,adssF2)
                                              (,adssF3 ,adssF4)
                                              (,adssF5)
                                              (,adssF6)
                                              (,adssF7)
                                              (,adssF8))))

(defun adssF1 ()
  ;; Dallas' defeat at the hand of Phoenix was its 27th straight at home.
 (setf adssF1
  `((deepsemcat entity)
    (concept game) (token pho-at-dal)
    (attrs ((results ((ents ((streak1 ((deepsemcat entity)
                                       (concept streak) (token dal-streak-vs-pho)
                                       (attrs ((card 27)))))
                             (streak1-gen-elt ((deepsemcat entity) (concept game)))))
                      (rels ((streak1-gen-elt ((deepsemcat relation)
                                               (role gen-elt) (token dal-streak-vs-pho-gen-elt)
                                               (args ((set {^4 ents streak1})
                                                      (gen-elt {^4 ents streak1-gen-elt})))))
                             (streak1-gen-elt-loser
                              ((deepsemcat relation)
                               (role loser) (token dal-streak-vs-pho-loser)
                               (args ((game {^4 ents streak1-gen-elt})
                                      (loser ((deepsemcat entity)
                                              (concept team) (token mavs)
                                              (attrs ((home "Dallas") (franchise "Maverick")))))))))
                             (streak1-gen-elt-host
                              ((deepsemcat relation)
                               (role host) (token dal-streak-vs-pho-host)
                               (args ((game {^4 ents streak1-gen-elt})
                                      (host ((deepsemcat entity)
                                             (concept team) (token mavs)
                                             (attrs ((home "Dallas") (franchise "Maverick")))))))))
                             (streak1-ext ((deepsemcat relation)
                                           (role streak-extension) (token dal-streak-vs-pho-ext)
                                           (args ((extension top-level)
                                                  (streak {^4 ents streak1}))))))))))))))
```

Figure C.4: List of floating fact lists in *Run 1*, with first sub-element

```
(defun adssF2 ()
  ;; This 27th straight home defeat by Dallas is a franchise record.
 (setf adssF2
  `((deepsemcat entity)
    (concept game) (token pho-at-dal)
    (attrs ((results
            ((ents ((histo-streak1 ((deepsemcat entity)
                                    (concept histo-streak) (token dal-streak-vs-pho-ref-set)))
                    (host-lifetime ((deepsemcat entity) (concept duration) (token dal-lifetime)))
                    (histo-streak1-gen-elt ((deepsemcat entity) (concept streak)))
                    (histo-streak1-gen-elt-gen-elt ((deepsemcat entity) (concept game)))
                    (histo-streak1-extr-card
                     ((deepsemcat entity)
                      (concept integer) (token dal-streak-vs-pho-ref-set-extr-card)))))
             (rels ((host-lifetime ((deepsemcat relation)
                                    (role lifetime) (token dal-lifetime-rel)
                                    (args ((entity ((deepsemcat entity)
                                                    (concept team) (token mavs)))
                                           (lifetime {^4 ents host-lifetime})))))
                    (histo-streak1-duration
                     ((deepsemcat relation)
                      (role duration) (token dal-streak-vs-pho-ref-set-duration-rel)
                      (args ((entity {^4 ents histo-streak1})
                             (duration {^4 ents host-lifetime})))))
                    (histo-streak1-gen-elt ((deepsemcat relation)
                                            (role gen-elt) (token dal-vs-pho-ref-set-gen-elt)
                                            (args ((set {^4 ents histo-streak1})
                                                   (gen-elt {^4 ents histo-streak1-gen-elt})))))
                    (histo-streak1-gen-elt-gen-elt
                     ((deepsemcat relation)
                      (role gen-elt)
                      (args ((set {^4 ents histo-streak1-gen-elt})
                             (gen-elt {^4 ents histo-streak1-gen-elt-gen-elt})))))
                    (histo-streak1-gen-elt-gen-elt-loser
                     ((deepsemcat relation)
                      (role loser) (token dal-vs-pho-ref-set-gen-elt-gen-elt-loser)
                      (args ((game {^4 ents histo-streak1-gen-elt-gen-elt})
                             (team ((deepsemcat entity) (concept team) (token mavs)))))))
                    (histo-streak1-gen-elt-gen-elt-host
                     ((deepsemcat relation)
                      (role host) (token dal-vs-pho-ref-set-gen-elt-gen-elt-host)
                      (args ((game {^4 ents histo-streak1-gen-elt-gen-elt})
                             (team ((deepsemcat entity) (concept team) (token mavs)))))))
                    (histo-streak1-extr-card ((deepsemcat relation)
                                              (role max-card)
                                              (token dal-streak-vs-pho-ref-set-extr-card-rel)
                                              (args ((card {^4 ents histo-streak1-extr-card})
                                                     (set {^4 ents histo-streak1})))))
                    (histo-streak1-update
                     ((deepsemcat relation)
                      (role >) (token dal-beat-franchise-record-streak-vs-pho)
                      (args ((streak-len 27)
                             (histo-streak-len-extr {^4 ents histo-streak1-extr-card}))))))))))))))
```

Figure C.5: Second floating fact in *Run 1*

```
(defun adssF3 ()
  ;; Danny Ainge scored 21 points
 (setf adssF3
  `((deepsemcat entity)
    (concept game) (token pho-at-dal)
    (attrs ((stats ((ents ((stat1-ca ((deepsemcat entity)
                                      (concept player) (token ainge)
                                      (attrs ((first-name "Danny") (last-name "Ainge")))))
                           (stat1 ((deepsemcat entity)
                                   (concept game-stat)
                                   (token ainge-pt-at-dal)
                                   (attrs ((value 21) (unit pt)))))))
                    (rels ((stat1 ((deepsemcat relation)
                                   (role game-stat-rel) (token ainge-scoring-at-dal)
                                   (args ((carrier {^4 ents stat1-ca})
                                          (stat {^4 ents stat1})))))))))))))

(defun adssF4 ()
  ;; Danny Ainge is a reserve player
 (setf adssF4
  `((deepsemcat entity)
    (concept game) (token pho-at-dal)
    (attrs ((stats ((ents ((stat1-ca-status ((deepsemcat entity)
                                             (concept reserve) (token ainge-reserve)))
                           (stat1-ca ((deepsemcat entity)
                                      (concept player) (token ainge)
                                      (attrs ((first-name "Danny") (last-name "Ainge")))))))
                    (rels ((stat1-ca-status ((deepsemcat relation)
                                             (role player-status) (token ainge-status)
                                             (args ((player {^4 ents stat1-ca})
                                                    (status {^4 ents stat1-ca-status})))))))))))))
```

Figure C.6: Third and fourth floating facts in *Run 1*

```
(defun adssF5 ()
  ;; The 42 points performance by Charles Barkley tied his season high.
 (setf adssF5
  `((deepsemcat entity)
    (concept game) (token pho-at-dal)
    (attrs ((stats ((ents ((histo-stat0 ((deepsemcat entity)
                                        (concept histo-stat) (token barkley-pt-at-dal-ref-set)))
                           (histo-stat0-duration ((deepsemcat entity)
                                                  (concept season)
                                                  (token barkley-pt-at-dal-ref-set-duration)))
                           (histo-stat0-gen-elt ((deepsemcat entity)
                                                 (concept game-stat)
                                                 (attrs ((unit pt)))))
                           (histo-stat0-gen-elt-ca ((deepsemcat entity)
                                                    (concept player) (token barkley)
                                                    (attrs ((first-name "Charles")
                                                            (last-name "Barkley")))))
                           (histo-stat0-extr ((deepsemcat entity)
                                              (concept integer)
                                              (token barkley-pt-at-dal-ref-set-extr)))))
                    (rels ((histo-stat0-duration  ((deepsemcat relation)
                                                   (role duration)
                                                   (token barkley-pt-at-dal-ref-set-duration-rel)
                                                   (args ((set {^4 ents histo-stat0})
                                                          (duration {^4 histo-stat0-duration})))))
                           (histo-stat0-gen-elt ((deepsemcat relation)
                                                 (role gen-elt)
                                                 (token barkley-pt-at-dal-ref-set-gen-elt)
                                                 (args ((set {^4 ents histo-stat0})
                                                        (gen-elt {^4 ents histo-stat0-gen-elt})))))
                           (histo-stat0-gen-elt-ca
                            ((deepsemcat relation)
                             (role game-stat-rel) (token barkley-pt-at-dal-ref-set-gen-elt-ca)
                             (args ((carrier {^4 ents histo-stat0-gen-elt-ca})
                                    (stat {^4 ents histo-stat0-gen-elt})))))
                           (histo-stat0-extr ((deepsemcat relation)
                                              (role max-val)
                                              (token barkley-pt-at-dal-ref-set-extr-rel)
                                              (args ((extr-val {^4 ents histo-stat0-extr})
                                                     (set {^4 ents histo-stat0})))))
                           (histo-stat0-update
                            ((deepsemcat relation)
                             (role =) (token barkley-tie-pt-season-high-at-dal)
                             (args ((stat-val 42)
                                    (histo-stat-extr {^4 ents histo-stat0-extr})))))))))))))
```

Figure C.7: Fifth floating fact in *Run 1*

```
(defun adssF6 ()
  ;; Ainge had 7 assists
 (setf adssF6
  `((deepsemcat entity)
    (concept game) (token pho-at-dal)
    (attrs ((stats ((ents ((stat2-ca ((deepsemcat entity)
                                      (concept player) (token ainge)
                                      (attrs ((first-name "Danny") (last-name "Ainge")))))
                           (stat2 ((deepsemcat entity)
                                   (concept game-stat) (token ainge-ast-at-dal)
                                   (attrs ((value 7) (unit ast)))))))
                    (rels ((stat2 ((deepsemcat relation)
                                   (role game-stat-rel) (token ainge-passing-at-dal)
                                   (args ((carrier {^4 ents stat2-ca})
                                          (stat {^4 ents stat2}))))))))))))))

(defun adssF7 ()
  ;; Barkley grabbed 10 rebounds
 (setf adssF7
  `((deepsemcat entity)
    (concept game) (token pho-at-dal)
    (attrs ((stats ((ents ((stat3-ca ((deepsemcat entity)
                                      (concept player) (token barkley)
                                      (attrs ((first-name "Charles") (last-name "Barkley")))))
                           (stat3 ((deepsemcat entity)
                                   (concept game-stat) (token barkley-reb-at-dal)
                                   (attrs ((value 10) (unit reb)))))))
                    (rels ((stat3 ((deepsemcat relation)
                                   (role game-stat-rel) (token barkley-rebounding-at-dal)
                                   (args ((carrier {^4 ents stat3-ca})
                                          (stat {^4 ents stat3}))))))))))))))

(defun adssF8 ()
  ;; Majerle scored 18 points
 (setf adssF8
  `((deepsemcat entity)
    (concept game) (token pho-at-dal)
    (attrs ((stats ((ents ((stat4-ca ((deepsemcat entity)
                                      (concept player) (token majerle)
                                      (attrs ((first-name "Dan") (last-name "Majerle")))))
                           (stat4 ((deepsemcat entity)
                                   (concept game-stat) (token majerle-pt-at-dal)
                                   (attrs ((value 18) (unit pt)))))))
                    (rels ((stat4 ((deepsemcat relation)
                                   (role game-stat-rel) (token majerle-scoring-at-dal)
                                   (args ((carrier {^4 ents stat4-ca})
                                          (stat {^4 ents stat4}))))))))))))))
```

Figure C.8: Sixth, seventh and eigth floating facts in *Run 1*

```
((deepsemcat entity)
 (concept game) (token pho-at-dal)
 (attrs ((setting ((ents ((addr ((deepsemcat entity)
                                (concept address) (token dallas-tx)
                                (attrs ((city "dallas") (state "tx")))))
                         (date ((deepsemcat entity)
                                (concept date) (token fri-nite)
                                (attrs ((day-name "friday") (day-part night)))))))))
        (stats ((ents ((stat0-ca ((deepsemcat entity)
                                   (concept player) (token barkley)
                                   (attrs ((first-name "charles") (last-name "barkley")))))
                        (stat0 ((deepsemcat entity) (concept game-stat) (token barkley-pt-at-dal)
                                (attrs ((value 42) (unit pt)))))
                        (stat1-ca ((deepsemcat entity)
                                   (concept player) (token ainge)
                                   (attrs ((first-name "danny") (last-name "ainge")))))
                        (stat1 ((deepsemcat entity)
                                (concept game-stat) (token ainge-pt-at-dal)
                                (attrs ((value 21) (unit pt)))))
                        (stat1-ca-status ((deepsemcat entity)
                                          (concept reserve) (token ainge-reserve)))
                        (histo-stat0 ((deepsemcat entity)
                                      (concept histo-stat) (token barkley-pt-at-dal-ref-set)))
                        (histo-stat0-duration ((deepsemcat entity)
                                               (concept season)
                                               (token barkley-pt-at-dal-ref-set-duration)))
                        (histo-stat0-gen-elt ((deepsemcat entity)
                                              (concept game-stat) (attrs ((unit pt)))))
                        (histo-stat0-gen-elt-ca ((deepsemcat entity)
                                                 (concept player) (token barkley)
                                                 (attrs ((first-name "charles")
                                                         (last-name "barkley")))))
                        (histo-stat0-extr ((deepsemcat entity)
                                           (concept integer)
                                           (token barkley-pt-at-dal-ref-set-extr)))))
```

Figure C.9: DSS layer of final draft in *Run 1*. Continued next page

```
(rels ((stat0 ((deepsemcat relation)
               (role game-stat-rel)
               (token barkley-scoring-at-dal)
               (args ((carrier {attrs stats ents stat0-ca})
                      (stat {attrs stats ents stat0})))))
       (stat1 ((deepsemcat relation)
               (role game-stat-rel) (token ainge-scoring-at-dal)
               (args ((carrier {attrs stats ents stat1-ca})
                      (stat {attrs stats ents stat1})))))
       (stat1-ca-status ((deepsemcat relation)
                         (role player-status) (token ainge-status)
                         (args ((player {attrs stats ents stat1-ca})
                                (status {attrs stats ents stat1-ca-status})))))
       (histo-stat0-duration
        ((deepsemcat relation)
         (role duration) (token barkley-pt-at-dal-ref-set-duration-rel)
         (args ((set {attrs stats ents histo-stat0})
                (duration {attrs stats histo-stat0-duration})))))
       (histo-stat0-gen-elt
        ((deepsemcat relation)
         (role gen-elt) (token barkley-pt-at-dal-ref-set-gen-elt)
         (args ((set {attrs stats ents histo-stat0})
                (gen-elt {attrs stats ents histo-stat0-gen-elt})))))
       (histo-stat0-gen-elt-ca
        ((deepsemcat relation)
         (role game-stat-rel) (token barkley-pt-at-dal-ref-set-gen-elt-ca)
         (args ((carrier {attrs stats ents histo-stat0-gen-elt-ca})
                (stat {attrs stats ents histo-stat0-gen-elt})))))
       (histo-stat0-extr ((deepsemcat relation)
                          (role max-val) (token barkley-pt-at-dal-ref-set-extr-rel)
                          (args ((extr-val {attrs stats ents histo-stat0-extr})
                                 (set {attrs stats ents histo-stat0})))))
       (histo-stat0-update
        ((deepsemcat relation)
         (role =) (token barkley-tie-pt-season-high-at-dal)
         (args ((stat-val 42)
                (histo-stat-extr {attrs stats ents histo-stat0-extr})))))))))
```

Figure C.9: DSS layer of final draft in *Run 1*. Continued from previous page and continued next page

```
(results ((ents ((host ((deepsemcat entity) (concept team) (token mavs)
                        (attrs ((home "dallas") (franchise "maverick")))))
                 (visitor ((deepsemcat entity) (concept team) (token suns)
                           (attrs ((home "phoenix") (franchise "sun")))))
                 (score ((deepsemcat entity) (concept score) (token pho-at-dal-score)
                         (attrs ((win 123) (lose 97)))))
                 (streak1 ((deepsemcat entity) (concept streak)
                           (token dal-streak-vs-pho) (attrs ((card 27)))))
                 (streak1-gen-elt ((deepsemcat entity) (concept game)))
                 (histo-streak1 ((deepsemcat entity) (concept histo-streak)
                                 (token dal-streak-vs-pho-ref-set)))
                 (host-lifetime ((deepsemcat entity) (concept duration)
                                 (token dal-lifetime)))
                 (histo-streak1-gen-elt ((deepsemcat entity) (concept streak)))
                 (histo-streak1-gen-elt-gen-elt ((deepsemcat entity) (concept game)))
                 (histo-streak1-extr-card
                  ((deepsemcat entity) (concept integer)
                   (token dal-streak-vs-pho-ref-set-extr-card)))))
          (rels ((winner ((deepsemcat relation)
                          (role winner) (token pho-at-dal-winner)
                          (args ((game top-level)
                                 (winner {attrs results ents visitor})))))
                 (loser ((deepsemcat relation)
                         (role loser) (token pho-at-dal-loser)
                         (args ((game top-level)
                                (loser {attrs results ents host})))))
                 (result ((deepsemcat relation)
                          (role beat) (token pho-at-dal-result)
                          (args ((winner {attrs results ents visitor})
                                 (loser {attrs results ents host})))))
                 (streak1-gen-elt
                  ((deepsemcat relation) (role gen-elt) (token dal-streak-vs-pho-gen-elt)
                   (args ((set {attrs results ents streak1})
                          (gen-elt {attrs results ents streak1-gen-elt})))))
                 (streak1-gen-elt-loser
                  ((deepsemcat relation) (role loser) (token dal-streak-vs-pho-loser)
                   (args ((game {attrs results ents streak1-gen-elt})
                          (loser ((deepsemcat entity) (concept team) (token mavs)
                                  (attrs ((home "dallas") (franchise "maverick")))))))))
                 (streak1-gen-elt-host
                  ((deepsemcat relation)
                   (role host) (token dal-streak-vs-pho-host)
                   (args ((game {attrs results ents streak1-gen-elt})
                          (host ((deepsemcat entity) (concept team) (token mavs)
                                 (attrs ((home "dallas") (franchise "maverick")))))))))
                 (streak1-ext ((deepsemcat relation)
                               (role streak-extension) (token dal-streak-vs-pho-ext)
                               (args ((extension top-level)
                                      (streak {attrs results ents streak1})))))
```

Figure C.9: DSS layer of final draft in *Run 1*. Continued from previous page and continued next page

```
(host-lifetime ((deepsemcat relation)
                (role lifetime) (token dal-lifetime-rel)
                (args ((entity ((deepsemcat entity)
                                (concept team) (token mavs)))
                       (lifetime {attrs results ents host-lifetime})))))
(histo-streak1-duration
 ((deepsemcat relation)
  (role duration) (token dal-streak-vs-pho-ref-set-duration-rel)
  (args ((entity {attrs results ents histo-streak1})
         (duration {attrs results ents host-lifetime})))))
(histo-streak1-gen-elt
 ((deepsemcat relation)
  (role gen-elt) (token dal-vs-pho-ref-set-gen-elt)
  (args ((set {attrs results ents histo-streak1})
         (gen-elt {attrs results ents histo-streak1-gen-elt})))))
(histo-streak1-gen-elt-gen-elt
 ((deepsemcat relation)
  (role gen-elt)
  (args ((set {attrs results ents histo-streak1-gen-elt})
         (gen-elt {attrs results ents histo-streak1-gen-elt-gen-elt})))))
(histo-streak1-gen-elt-gen-elt-loser
 ((deepsemcat relation)
  (role loser) (token dal-vs-pho-ref-set-gen-elt-gen-elt-loser)
  (args ((game {attrs results ents histo-streak1-gen-elt-gen-elt})
         (team ((deepsemcat entity) (concept team) (token mavs)))))))
(histo-streak1-gen-elt-gen-elt-host
 ((deepsemcat relation)
  (role host) (token dal-vs-pho-ref-set-gen-elt-gen-elt-host)
  (args ((game {attrs results ents histo-streak1-gen-elt-gen-elt})
         (team ((deepsemcat entity) (concept team) (token mavs)))))))
(histo-streak1-extr-card
 ((deepsemcat relation) (role max-card)
  (token dal-streak-vs-pho-ref-set-extr-card-rel)
  (args ((card {attrs results ents histo-streak1-extr-card})
         (set {attrs results ents histo-streak1})))))
(histo-streak1-update
 ((deepsemcat relation)
  (role >) (token dal-beat-franchise-record-streak-vs-pho)
  (args ((streak-len 27)
         (histo-streak-len-extr
          {attrs results ents histo-streak1-extr-card})))))))))))
```

Figure C.9: DSS layer of final draft in *Run 1.* Continued from previous page

```
((surfsemcat rhetor) (struct hypotax)
 (root ((surfsemcat rhetor) (struct paratax) (rel teammate)
       (elts ((car ((surfsemcat rhetor) (struct hypotax)
                    (root ((surfsemcat encyclo) (onto event) (concept =)
                           (args ((agent ((surfsemcat encyclo) (onto indiv) (concept player)
                                          (names ((first-name "charles") (last-name "barkley")))))
                                  (affected ((surfsemcat rhetor) (struct hypotax)
                                             (root ((surfsemcat encyclo)
                                                    (concept max-val) (onto indiv)))
                                             (rels ((duration ((surfsemcat encyclo)
                                                               (onto indiv)
                                                               (concept season))))))))))
                    (rels ((instrument ((surfsemcat encyclo) (onto quantity) (concept game-stat)
                                        (restrictors ((value 42) (unit pt)))))))))
              (cdr ((car ((surfsemcat rhetor) (struct hypotax)
                          (root ((surfsemcat rhetor) (struct event) (root transf-loc)
                                 (args ((agent ((surfsemcat encyclo) (onto indiv) (concept player)
                                                (names ((first-name "danny")
                                                        (last-name "ainge")))))
                                        (located {root args agent})
                                        (location ((surfsemcat encyclo)
                                                   (onto place) (concept reserve)))))))
                          (rels ((result ((surfsemcat encyclo) (onto event) (concept game-stat-rel)
                                          (args ((agent ((surfsemcat encyclo) (onto indiv)
                                                         (concept {root args agent concept})
                                                         (names {root args agent names})))
                                                 (created ((surfsemcat encyclo)
                                                           (onto quantity) (concept game-stat)
                                                           (restrictors ((value 21)
                                                                         (unit pt))))))))))))
                    (cdr none)))))))
 (rels ((location ((surfsemcat encyclo) (onto place) (concept address)
                   (restrictors ((city "dallas") (state "tx"))))))
```

Figure C.10: SSS layer of final draft in *Run 1*. Continued next page

```
(time ((surfsemcat rhetor) (struct paratax) (rel temporal-inclusion)
       (elts ((car ((surfsemcat encyclo) (concept date) (onto time)
                    (restrictors ((day-name "friday") (day-part night)))))
              (cdr ((car ((surfsemcat rhetor) (struct hypotax)
                          (root
                           ((surfsemcat rhetor)
                            (struct event) (root transf-poss)
                            (args ((agent ((surfsemcat encyclo)
                                           (onto indiv) (concept team) (ref full)
                                           (names ((home "phoenix")
                                                   (franchise "sun")))))
                                   (affected ((surfsemcat encyclo)
                                              (onto indiv) (concept team) (ref full)
                                              (names ((home "dallas")
                                                      (franchise "maverick")))))
                                   (possessed
                                    ((surfsemcat rhetor) (struct hypotax)
                                     (root
                                      ((surfsemcat rhetor)
                                       (struct hypotax)
                                       (root ((surfsemcat encyclo)
                                              (onto indiv) (concept loser)))
                                       (rels ((possessor full-ref)
                                              (ordinal
                                               ((surfsemcat rhetor) (struct hypotax)
                                                (root ((surfsemcat encyclo)
                                                       (onto quantity)
                                                       (concept integer)
                                                       (restrictors ((value 27)))))
                                                (rels ((card-of ((surfsemcat encyclo)
                                                                 (onto place)
                                                                 (concept streak)))))))

                                              (location ((surfsemcat encyclo)
                                                         (onto place) (concept host)))
                                              (compar
                                               ((surfsemcat rhetor)
                                                (struct hypotax)
                                                (root ((surfsemcat encyclo)
                                                       (onto indiv) (concept max-card)))
                                                (rels ((duration
                                                        ((surfsemcat encyclo)
                                                         (onto indiv)
                                                         (concept lifetime)
                                                         (restrictors
                                                          ((scope
                                                            ((surfsemcat encyclo)
                                                             (onto indiv)
                                                             (concept team)
                                                             (ref empty))))))))))))))))))))
                          (rels ((score ((surfsemcat encyclo)
                                         (onto quantity) (concept score)
                                         (restrictors ((win 123) (lose 97)))))))))
                 (cdr none)))))))))
```

Figure C.10: SSS layer of final draft in *Run 1*. Continued from previous page

```
((cat clause) (complex conjunction)
 (distinct ((car ((cat clause) (tense past)
                  (proc ((type material) (lex "tie")))
                  (partic ((agent ((cat compound-proper)
                                   (head ((cat person-name)
                                          (first-name ((lex "charles")))
                                          (last-name ((lex "barkley")))))))
                           (affected ((cat common)
                                      (classifier ((cat noun)
                                                   (synt-funct classifier) (lex "season")))
                                      (possessor ((cat personal-pronoun) (gender masculine)))
                                      (head ((lex "record")))))))
                  (pred-modif ((instrument ((cat pp)
                                            (np ((cat measure)
                                                 (quantity ((value 42)))
                                                 (unit ((lex "point")))))))))))
            (cdr ((car ((cat clause) (tense past)
                        (proc ((type composite) (relation-type locative)
                               (effective no) (lex "come")))
                        (partic ((agent ((cat compound-proper)
                                         (head ((cat person-name)
                                                (first-name ((lex "danny")))
                                                (last-name ((lex "ainge")))))))
                                 (located {distinct cdr car partic agent})
                                 (location ((cat pp) (prep ((lex "off")))
                                            (np ((cat common) (lex "bench")))))))
                        (circum
                         ((result ((cat clause) (tense past) (mood to-infinitive)
                                   (proc ((type material) (effect-type creative) (lex "add")))
                                   (partic ((agent {distinct cdr car circum result controlled})
                                            (created ((cat measure)
                                                      (quantity ((value 21)))
                                                      (unit ((gap yes) (lex "point")))))))
                                    (punctuation ((before none))))))))))))
```

Figure C.11: DGS layer of final draft in *Run 1.* Continued next page

```
(circum ((location ((cat address) (city ((lex "dallas"))) (state ((lex "tx"))) (position header)))
         (time ((cat list)
                (distinct
                 ((car ((cat date) (day-name ((lex "friday"))) (day-part ((lex night)))))
                  (cdr ((car ((cat clause) (tense past)
                              (mood bound-adverbial) (position end) (binder ((lex "as")))
                              (proc ((type composite) (relation-type possessive) (lex "give")))
                              (partic ((agent ((cat compound-proper)
                                               (head ((cat team-name)
                                                      (home ((lex "phoenix")))
                                                      (franchise ((lex "sun")))))
                                               (number plural)))
                                       (affected ((cat compound-proper)
                                                  (head ((cat team-name)
                                                         (home ((lex "dallas")))
                                                         (franchise ((lex "maverick")))))
                                                  (number plural)))
                                       (possessor {circum time distinct cdr car partic affected})
                                       (possessed ((cat common)
                                                   (possessor ((cat personal-pronoun)
                                                               (number plural)))
                                                   (head ((cat measure)
                                                          (quantity ((cat compound-ordinal)
                                                                     (numeral ((cat ordinal)
                                                                               (value 27)))
                                                                     (complement
                                                                      ((cat pp)
                                                                       (prep ((lex "in")))
                                                                       (np ((cat common)
                                                                            (definite no)
                                                                            (lex "row")))))))
                                                          (unit ((lex "defeat")))))
                                                   (qualifier
                                                    ((cat pp)
                                                     (prep ((lex "at")))
                                                     (np ((cat common)
                                                          (determiner none) (lex "home")))))
                                                   (classifier
                                                    ((cat noun-compound)
                                                     (synt-funct classifier)
                                                     (classifier ((cat noun)
                                                                  (synt-funct classifier)
                                                                  (lex "franchise")))
                                                     (head ((lex "worst")))))))))
                              (pred-modif ((score ((cat score)
                                                   (win ((value 123))) (lose ((value 97)))))))))
                        (cdr none))))))))
```

Figure C.11: DGS layer of final draft in *Run 1*. Continued from previous page fact to *Draft 2*, STREAK uses the monotonic revision rule `Coordinative Conjoin of Clause`. This rule was applied to the main statistic clause, "*Charles Barkley registered 42 points*", because this additional statistic also concerns a player of the winning team. Furthermore, since they are both scoring performances, STREAK exploits this fact and chooses to elide the head of the object in the added conjoined clause (resulting in the phrase "*Danny Ainge added 24 ∅*" instead of "*Danny Ainge added 24 points*"). This illustrates how STREAK opportunistically takes advantage of the particular draft context into which a floating fact is woven, to choose a more concise expression for that fact. STREAK also uses this context to make the most appropriate lexical choice, as illustrated by the choice of the verb "*to add*" for this second statistic. Such a verb can be chosen only in this particular context. It would be inappropriate for example, to realize the main statistic, for which STREAK chose the more general verb "*to register*" in this particular run. The code for the choice of such verbs was given in Section 4.3.2. How the reviser passes such contextual information to the lexicalizer was also explained in that section.

The fourth sub-element in `float-stack-F` is non-historical fact of type player status. Just as the second floating fact underlined the significance of the first by conveying its record breaking nature, this fourth fact underlines the significance of the third. It notes that the player whose scoring statistic was just added to the draft ("*Danny Ainge added 24*"), is a reserve player (making the fact that he scored that many points all the more remarkable). To add this fact to *Draft 3*, STREAK uses the monotonic revision rule `Absorb of Clause into Clause as Result`. Moreover, it uses the specialization of this revision rule that involves the side transformation `Agent Control`. This specialization is chosen when STREAK notices that both the absorbed and absorbing clauses share the same agent. It allows the agent of the absorbed one, which was part of the original draft, to be deleted, resulting in "*Danny Ainge came off the bench ∅ to add 24*" instead of "*Danny Ainge came off the bench for Danny Ainge to add 24*", thus opportunistically gaining space and fluency. This example illustrates the capability of STREAK to use idioms such as the expression "*to come off the bench*" which conveys that a player is a reserve.

The fifth sub-element in `float-stack-F` is a historical fact of type record equalling. It concerns the main statistic. To add this fact to *Draft 4*, STREAK uses the revision rule `Adjunctization of Created into Instrument`. It moves the object of the main statistic clause that was filling the `Created` role in that clause[2], to an `Instrument` role in order to accommodate the added record as object. The equalling aspect of this record is expressed as the new main verb "to match" replacing the original verb "*to register*". The action explicitly conveyed by this original verb is now implicitly conveyed by "*to match*", since matching a record can only come as a consequence of a performance. This example thus illustrates the ability of STREAK to opportunistically take advantage of the addition of a new fact to gain space by making implicit part of the realization of another fact already in the draft. It also demonstrates how STREAK takes into account stylistic conventions observed in the corpus. Compare the addition of this fifth floating fact with the addition of the second one. They both concern a record, the difference between them being that the second fact expresses that a record was *broken* and the fifth one that it was merely *equalled*. This difference, which could seem minor at first, triggers the use of entirely different revision rules: the monotonic `Adjoin` for the second fact and the non-monotonic `Adjunctization` for the fifth one. This difference in strategies implements the stylistic convention observed among sports or stock market writers that mentioning of a record update event without explicitly specifying whether it is of the breaking or equalling type implies that it is of the breaking type. This convention allows STREAK to use the simple and concise revision rule `Adjoin` for record breaking events: note how nothing in *Draft 2* specifies whether the 27th defeat of Dallas actually breaks or merely equals their longest losing streak. Using such an implicit form for record equalling events as well would be misguiding however. The need to keep reports concise must be balanced with the need to keep these two type of events unambiguously distinguishable. It is in order to be explicit about the equalling type of the record update event added in the fifth increment, that STREAK uses the less concise and more complex `Adjunctization` revision.

After the addition of this fifth floating fact, the draft is only two words away from the maximum length of 45 observed in the corpus. Thus, unless the next sub-element in `float-stack-F` can be added with only two more words, it will not fit in this lead sentence summary. This next sub-element is an additional statistic, the passing performance of Danny Ainge. The most concise way it can be accommodated in the

[2] cf. sections B.2.2.1 and B.3.2 of Appendix B for the definition of the thematic roles used in STREAK.

draft is by revising the nominal realizing the scoring statistic of this player that was added during the third revision increment and which is already reduced to the cardinal number "*21*". STREAK applies the revision rule `Coordinative Conjoin of Nominal` to this nominal, yielding "*to add 21* **and 7 assists**". This revision thus adds three new words (in bold) while not deleting any and thus pushes the revised draft over the length limit. STREAK thus halts the revision process without printing the draft resulting from this final revision. As final value, the `revise` function returns the three-layer FD representing the previous draft, which was under the complexity limits. Since this FD is very large, its body is not shown in Fig. C.1, but its presence is signaled by the abbreviation `((SSS ((DSS ... ))))`. The full detailed body for this final draft representation is given in three figures, each extending over several pages. The DSS layer of the draft in shown in Fig. C.9, its SSS layer in Fig. C.10 and its DGS layer in Fig. C.11. For the sake of legibility, the values in an outer layer that are inherited from an inner layer are shown as copies in this series of figures. However, in the actual draft representation generated and then revised by the system these values are really pointers (from the outer layer to the inner layer). For example, the value of under the path `{circum time distinct cdr car pred-modif score win value}` is the atom `123` in the DGS of Fig. C.11. In fact the actual representation generated by STREAK, value is really the path `{attr results ents score attrs win}` which points to the winning team score in the DSS of Fig. C.9.

C.1.2 Example run 2: Choice of revision rule constrained by the surface form of the draft

Run 2 shown in Fig. C.12 illustrates how in STREAK, the choice of a revision rule to add a given floating fact onto the draft is sensitive not only to the content of the draft, but to its surface form as well. It starts with two calls to the function `draft` to build two alternative draft forms, from the same input but a with different realization constraint. *Run 2* was already presented in Section C.1.1. In this section, I show the output again, but this time I also give:

- The three-layer internal representation of the two synonymous drafts which trigger the different revision rules.
- The pre-condition of the `Nominalization` revision rule, showing why it gets triggered for one of the two drafts and not the other.

The common input to these two revision stages is a DSS FD encoding the four fixed facts to convey in both draft. It is built by calling the function `dssC0` and shown in Fig. C.14. For the first call to `draft`, the realization constraint is `(form-flag1)` whose value was given in Fig. C.2. It constrains the output `draftC1` to express the game result as a *full* verb clause that follows the pattern "WINNER full-verb LOSER" and is subordinated to the main statistic clause as a time adjunct. In this particular run, the full verb chosen (randomly) by the lexicalizer is "*to beat*".

For the second call, the realization constraint is `(form-flag2)` whose value, also given in Fig. C.2, constrains the output `draftC2` to express the game result as a *support* verb clause following the pattern "WINNER support-verb LOSER nominal"[3]. In this particular run the support-verb/nominal-head collocation chosen (randomly) by the lexicalizer is "*to nail down a win*".

Once these two alternative synonymous draft sentences have been built, the function `revise1` is then called on each of them with the same additional floating fact `adssC4` as second parameter. `revise1` is the function to call for a single revision increment. It implements the revision rule interpreter described in Section 4.3.3.2. In contrast, the function `revise` called for *Run 1* is used for chaining revisions and implements the revision monitor described in Section 4.3.3.2.3. `revise` works by traversing the list of lists of floating facts and repeatedly calling `revise1` on each fact. `adssC4` encodes a losing streak extension for the Boston Celtics. To incorporate this floating fact to `draftC0`, STREAK uses the revision rule `Nominalization`. In contrast, to incorporate this same floating fact on the synonymous but linguistically distinct `draftC1`, STREAK uses the revision rule `Adjunctization`. In each case, the choice of one revision rule over the other

[3] Like `(form-flag1)`, `(form-flag2)` also constrains the game result clause to be subordinated to the main statistic clause as a time adjunct.

```
> (setf draftC1 (draft (dssC0) :sss (form-flag1)))  ;; Draft form 1

Hartford, CT -- Karl Malone hit for 39 points Friday night as the Utah Jazz beat the
Boston Celtics 98 - 94.

((SSS ((DSS .... ))))
> (setf draftC2 (draft (dssC0) :sss (form-flag2)))  ;; Draft form 2

Hartford, CT -- Karl Malone had 39 points Friday night as the Utah Jazz nailed down a
98 - 94 win over the Boston Celtics.

((SSS ((DSS .... ))))
> (revise1 draftC1 (adssC4))  ;; Revising Draft 1 using Nominalization

Hartford, CT -- Karl Malone hit for 39 points Friday night as the Utah Jazz brought the
Boston Celtics their sixth consecutive setback at home  98 - 94.

((SSS ((DSS .... ))))
> (revise1 draftC2 (adssC4))  ;; Revising Draft 2 using Adjunctization

Hartford, CT -- Karl Malone had 39 points Friday night as the Utah Jazz extended the
Celtics' homecourt losing streak to six with a 98 - 94 win  over Boston.

((SSS ((DSS .... ))))
>
```

Figure C.12: STREAK using different revision rules depending on the surface form of the draft

```
(def-conj nominalization
  (lhs ((bls ((:& material-basic-res-cluster)))
        (adss ((attrs ((results ((:& los-streak-ext)))))))
        (tool nominalization)))
  (rhs ( ...

(def-conj material-basic-res-cluster
  (cat #(under clause))
  (partic ((agent ((sss {^3 sss root args agent})))
           (affected ((sss {^3 sss root args affected})))))
  (pred-modif ((score ((sss {^3 sss rels score})))))
  (sss ((surfsemcat #(under rhetor))
        (struct #(under hypotax))
        (root ((surfsemcat #(under encyclo))
               (args ((agent ((surfsemcat #(under encyclo))
                              (dss {^3 dss args winner})))
                      (affected ((surfsemcat #(under encyclo))
                                 (dss {^3 dss args loser})))))
               (dss ((deepsemcat #(under relation))
                     (role #(under beat))
                     (args ((winner ((deepsemcat #(under entity))
                                     (concept #(under team))))
                            (loser ((deepsemcat #(under entity))
                                    (concept #(under team))))))))))
        (rels ((score ((surfsemcat #(under encyclo))
                       (dss ((deepsemcat #(under entity))
                             (concept #(under score))))))))))

(def-conj los-streak-ext
  (:& streak-ext)
  (rels ((streak1-gen-elt-loser ((deepsemcat #(under relation))
                                 (role #(under loser)))))))

(def-conj streak-ext
  (ents ((streak1 ((:& streak)))
         (streak1-gen-elt ((deepsemcat #(under entity))
                           (concept #(under game))))))
  (rels ((streak1-gen-elt ((deepsemcat #(under relation))
                           (role #(under gen-elt))
                           (args ((set {^4 ents streak1})
                                  (gen-elt {^4 ents streak1-gen-elt})))))
         (streak1-ext ((deepsemcat #(under relation))
                       (role #(under streak-extension))
                       (args ((extension #(under top-level))
                              (streak {^4 ents streak1}))))))))

(def-conj streak (deepsemcat #(under entity))
                 (concept #(under streak))
                 (attrs ((card GIVEN))))
```

Figure C.13: Pre-conditions to using nominalization for losing streak extensions

```
((deepsemcat entity)
 (concept game) (token uta-at-bos)
 (attrs ((setting ((ents ((addr ((deepsemcat entity)
                                 (concept address) (token hartford-ct)
                                 (attrs ((city "hartford") (state "ct")))))
                          (date ((deepsemcat entity)
                                 (concept date) (token fri-nite)
                                 (attrs ((day-name "friday") (day-part night)))))))))
        (stats ((ents ((stat0-ca ((deepsemcat entity)
                                    (concept player) (token kmalone)
                                    (attrs ((first-name "karl") (last-name "malone")))))
                        (stat0 ((deepsemcat entity)
                                 (concept game-stat) (token kmalone-pt-at-bos)
                                 (attrs ((value 39) (unit pt)))))))
                (rels ((stat0 ((deepsemcat relation)
                                 (role game-stat-rel) (token kmalone-scoring-at-bos)
                                 (args ((carrier {sss dss attrs stats ents stat0-ca})
                                        (stat {sss dss attrs stats ents stat0})))))))))
        (results ((ents ((host ((DEEPSEMCAT entity)
                                  (CONCEPT team) (token celts)
                                  (attrs ((home "boston") (franchise "celtic")))))
                          (visitor ((DEEPSEMCAT entity)
                                     (CONCEPT team) (token jazz)
                                     (attrs ((home "utah") (franchise "jazz")))))
                          (score ((DEEPSEMCAT entity)
                                   (CONCEPT score) (token uta-at-bos-score)
                                   (attrs ((win 98) (lose 94)))))))
                  (rels ((winner ((deepsemcat relation)
                                    (role winner) (token uta-at-bos-winner)
                                    (args ((game top-level)
                                           (winner {sss dss attrs results ents visitor})))))
                          (loser ((deepsemcat relation)
                                   (role loser) (token uta-at-bos-loser)
                                   (args ((game top-level)
                                          (loser {sss dss attrs results ents host})))))
                          (result ((DEEPSEMCAT relation)
                                    (ROLE beat) (token uta-at-bos-result)
                                    (ARGS ((WINNER {SSS DSS ATTRS RESULTS ENTS VISITOR})
                                           (LOSER {SSS DSS ATTRS RESULTS ENTS HOST})))))))))))
```

Figure C.14: DSS layer of both drafts in *Run 2*

```
((surfsemcat rhetor)
 (struct hypotax)
 (root ((surfsemcat encyclo)
        (onto event) (concept game-stat-rel)
        (args ((agent ((surfsemcat encyclo)
                       (onto indiv) (concept player)
                       (names ((first-name "karl") (last-name "malone")))))
               (created ((surfsemcat encyclo)
                         (concept game-stat) (onto quantity)
                         (restrictors ((value 39) (unit pt))))))))
 (rels ((time ((surfsemcat rhetor)
               (struct paratax) (rel temporal-inclusion)
               (elts ((car ((surfsemcat encyclo)
                            (concept date) (onto time)
                            (restrictors ((day-name "friday") (day-part night)))))
                      (cdr ((car ((SURFSEMCAT rhetor)
                                  (STRUCT hypotax)
                                  (ROOT ((SURFSEMCAT encyclo)
                                         (onto event) (concept beat)
                                         (ARGS ((AGENT ((SURFSEMCAT encyclo)
                                                        (onto indiv) (concept team) (ref full)
                                                        (names ((home "utah") (franchise "jazz")))))
                                                (AFFECTED ((SURFSEMCAT encyclo)
                                                           (onto indiv) (concept team)
                                                           (ref full)
                                                           (names ((home "boston")
                                                                   (franchise "celtic")))))))))
                                  (RELS ((SCORE ((SURFSEMCAT encyclo)
                                                 (ONTO quantity) (concept score)
                                                 (restrictors ((win 98) (lose 94))))))))
                            (cdr none)))))))
        (location ((surfsemcat encyclo)
                   (concept address) (onto place)
                   (restrictors ((city "hartford") (state "ct"))))))))
```

Figure C.15: SSS layer of the first initial draft in *Run 2*

```
((cat clause)
 (tense past)
 (proc ((type material) (effect-type creative) (lex "hit for")))
 (partic ((agent ((cat compound-proper)
                  (head ((cat person-name)
                         (first-name ((lex "karl"))) (last-name ((lex "malone")))))))
         (created ((cat measure)
                   (quantity ((value 39))) (unit ((lex "point")))))))
 (circum ((location ((position header) (cat address) (city ((lex "hartford")))
                     (state ((lex "ct")))))
         (time ((cat list)
                (fills time-rel)
                (distinct ((car ((cat date)
                                 (day-name ((lex "friday"))) (day-part ((lex night)))))
                           (cdr ((car ((CAT clause)
                                       (tense past) (mood bound-adverbial)
                                       (position end) (binder ((lex "as")))
                                       (proc ((type material) (lex "beat")))
                                       (PARTIC ((AGENT ((cat compound-proper)
                                                        (head ((cat team-name)
                                                               (home ((lex "utah")))
                                                               (franchise ((lex "jazz")))))
                                                        (number plural)))
                                                (AFFECTED ((cat compound-proper)
                                                           (head ((cat team-name)
                                                                  (home ((lex "boston")))
                                                                  (franchise ((lex "celtic")))))
                                                           (number plural)))))
                                       (PRED-MODIF ((SCORE ((cat score)
                                                            (win ((value 98)))
                                                            (lose ((value 94))))))))))
                                 (cdr none))))))))))
```

Figure C.16: DGS layer of the first initial draft in *Run 2*

```
((surfsemcat rhetor)
 (struct hypotax)
 (root ((surfsemcat encyclo)
        (onto event) (concept game-stat-rel)
        (args ((agent ((surfsemcat encyclo)
                       (onto indiv) (concept player)
                       (names ((first-name "karl") (last-name "malone")))))
               (created ((surfsemcat encyclo)
                         (onto quantity) (concept game-stat)
                         (restrictors ((value 39) (unit pt))))))))))
 (rels ((location ((surfsemcat encyclo)
                   (concept address) (onto place)
                   (restrictors ((city "hartford") (state "ct")))))
        (time ((surfsemcat rhetor)
               (struct paratax) (rel temporal-inclusion)
               (elts ((car ((surfsemcat encyclo)
                            (concept date) (onto time)
                            (restrictors ((day-name "friday") (day-part night)))))
                      (cdr ((car ((SURFSEMCAT rhetor)
                                  (STRUCT event) (ROOT activity)
                                  (args ((agent ((surfsemcat encyclo)
                                                 (onto indiv) (concept team) (ref full)
                                                 (names ((home "utah") (franchise "jazz")))))

                                         (range ((surfsemcat rhetor)
                                                 (struct hypotax) (already-mapped winner)
                                                 (root ((surfsemcat encyclo)
                                                        (concept winner) (onto indiv)))
                                                 (rels ((score ((surfsemcat encyclo)
                                                                (concept score) (onto quantity)
                                                                (restrictors ((win 98)
                                                                              (lose 94)))))
                                                        (opposition
                                                         ((surfsemcat encyclo)
                                                          (onto indiv)
                                                          (concept team)
                                                          (ref full)
                                                          (names ((home "boston")
                                                                  (franchise "celtic"))))))))))))
                            (cdr none)))))))))
```

Figure C.17: SSS layer of the second initial draft in *Run 2*

```
((cat clause)
 (tense past)
 (proc ((type material) (effect-type creative) (lex "have")))
 (partic ((agent ((cat compound-proper)
                  (head ((cat person-name)
                         (first-name ((lex "karl"))) (last-name ((lex "malone")))))))
          (created ((cat measure)
                    (quantity ((value 39))) (unit ((lex "point")))))))
 (circum ((location ((cat address)
                     (position header) (city ((lex "hartford"))) (state ((lex "ct")))))
          (time ((cat list)
                 (distinct ((car ((cat date)
                                  (day-name ((lex "friday"))) (day-part ((lex night)))))
                            (cdr ((car ((CAT clause)
                                        (tense past) (mood bound-adverbial)
                                        (position end) (binder ((lex "as")))
                                        (proc ((type material) (effective no) (lex "nail down")))
                                        (PARTIC ((AGENT ((cat compound-proper)
                                                         (head ((cat team-name)
                                                                (home ((lex "utah")))
                                                                (franchise ((lex "jazz")))))
                                                         (number plural)))
                                                 (range ((cat common)
                                                         (definite no)
                                                         (classifier ((cat score)
                                                                      (win ((value 98)))
                                                                      (lose ((value 94)))))
                                                         (qualifier
                                                          ((cat pp)
                                                           (prep ((lex "over")))
                                                           (np ((cat compound-proper)
                                                                (head ((cat team-name)
                                                                       (home ((lex "boston")))
                                                                       (franchise ((lex "celtic")))))
                                                                (number plural)))))
                                                         (head ((lex "win")))))))))
                                  (cdr none)))))))))
```

Figure C.18: DGS layer of the second initial draft in *Run 2* is motivated by the surface form of the respective drafts involved. `Nominalization` realizes a new fact by modifiers attached to a nominal resulting from the transformation of a full-verb clause into a support-verb clause. `Adjunctization` conversely replaces a support-verb clause by a full-verb clause incorporating the new fact by a full-verb with a new object while displacing the original object to an adjunct position. Since `draftC1` follows a full-verb pattern, only `Nominalization` and not `Adjunctization` is applicable to it. For `draftC2` following a support verb pattern, it is just the opposite. There is no game result NP in `draftC1` to be adjunctized and no game result full verb in `draftC2` to be nominalized. It is precisely because the applicability of revision rules such as the two above is dependent on surface form that the presence of the DGS layer in the draft representation of STREAK is required.

The surface form constraints in the pre-condition of `Nominalization` are given in Fig. C.13. The common DSS for both `draftC1` and `draftC2` is given in Fig. C.14. The SSS of each draft form are respectively given in figures C.15 and C.17. and the corresponding DGS in figures C.16 and C.18. In each of these FDs, the attributes that are tested by the pre-condition of the `nominalization` revision rules are uppercased. An example attribute of `draftC1` whose value matches a pre-condition to the application of `Nominalization` is the attribute `struct` under the path `{rels time elts cdr car}` in Fig. C.15 containing the SSS layer of `draftC1`. Its value, `hypotax` matches that of the corresponding attribute of the `Nominalization` pre-condition (shown on line 13 in Fig. C.13). Now consider the same attribute in Fig. C.17 containing the SSS layer of `draftC2` . Its value is `event`, thus causing the unification of `draftC2` with the pre-condition of `Nominalization` to fail. This example also illustrates the structure traversal of the draft's DGS and SSS layers that the revision rule interpreter performs while trying to apply a given revision rule. Note how the match for `draftC1` and mismatch for `draftC2` just discussed occur when the SSS part of pre-condition for the `Nominalization` revision rule is unified with *the sub-FD under the path* `{rels time elts cdr car}` in the SSS layer of the draft. This unification is attempted only after failure to unify the pre-condition of the revision rule[4] with any embedding sub-FD.

C.2 Additional example runs

In the following sections I present three additional example runs describing other interesting aspects of STREAK. The first of these additional runs, *Run 3*, consists of draft building stages that illustrate the paraphrasing power encoded in the phrase planner and the lexicalizer. It shows a variety of draft forms generated from the same set of input fixed facts. The next additional run, *Run 4* consists of parallel revision increments that illustrates the paraphrasing power encoded in the reviser. It shows how the same floating fact can be incorporated into the same initial draft form by using different revision rules resulting in a variety of revised draft forms. The last additional run, *Run 5* provides another example incremental complex sentence generation. Whereas *Run 1* illustrated the variety of revision rules by applying a different rule at each increment, *Run 5* illustrates the applicability of the same rules in different textual contexts by repeatedly applying the same rules at a different levels inside the draft structure.

C.2.1 Example run 3: Generating draft paraphrases

As explained in Section 7.2.2.2, there are multiple sources of paraphrasing power in STREAK. There are two such sources at play at draft time: alternative phrase planning rules and alternative lexicalization rules. *Run 3*, shown in Fig. C.19 illustrates the effects of these two sources. It contains 13 draft building stages from the same semantic input. This input, generated by a call to the function `dssE0` is similar to the draft stage input `dssF0` of *Run 1* that was shown in Fig. C.3. It contains exactly the same concepts only different instances of each.

Each draft in *Run 3* is built by calling the function `draft`. Paraphrases 1 to 8 are produced by constraining the syntagmatic structure of the draft using the keyword parameter `:sss` with one of the three form flags given in Fig. C.2. `form-flag1`, used in paraphrases 1, 4 and 7, forces STREAK to choose an hypotactic top-level structure where the main statistic is the matrix clause and the game result is a dependent clause.

[4] Or any revision rule depending whether a specific revision rule is indicated in the input to the reviser.

```
> (progn (draft (dssE0) :sss (form-flag1)) (values))            ;; Paraphrase 1
Orlando, FL -- Shaquille O'Neal rattled off 37 points Friday night as
the Orlando Magic beat the Toronto Raptors 101 - 89.

> (progn (draft (dssE0) :sss (form-flag2)) (values))            ;; Paraphrase 2
Orlando, FL -- Shaquille O'Neal tossed in 37 points Friday night as the
Orlando Magic posted a 101 - 89 triumph against the Toronto Raptors.

> (progn (draft (dssE0) :sss (form-flag3)) (values))            ;; Paraphrase 3
Orlando, FL -- Shaquille O'Neal chipped in 37 points Friday night,
rallying the Orlando Magic to a 101 - 89 win over the Toronto Raptors.

> (progn (draft (dssE0) :sss (form-flag1)) (values))            ;; Paraphrase 4
Orlando, FL -- Shaquille O'Neal hit for 37 points Friday night while the
Orlando Magic routed the Toronto Raptors 101 - 89.

> (progn (draft (dssE0) :sss (form-flag2)) (values))            ;; Paraphrase 5
Orlando, FL -- Shaquille O'Neal tallied 37 points Friday night while the
Orlando Magic clinched a 101 - 89 victory over the Toronto Raptors.

> (progn (draft (dssE0) :sss (form-flag3)) (values))            ;; Paraphrase 6
Orlando, FL -- Shaquille O'Neal finished with 37 points Friday night,
pacing the Orlando Magic to a 101 - 89 win over the Toronto Raptors.

> (progn (draft (dssE0) :sss (form-flag1)) (values))            ;; Paraphrase 7
Orlando, FL -- Shaquille O'Neal stroke for 37 points Friday night as the
Orlando Magic triumphed over the Toronto Raptors 101 - 89.

> (progn (draft (dssE0) :sss (form-flag2)) (values))            ;; Paraphrase 8
Orlando, FL -- Shaquille O'Neal fired in 37 points Friday night as the
Orlando Magic recorded a 101 - 89 victory over the Toronto Raptors.

> (progn (draft (dssE0) :sss (form-flag3)) (values))            ;; Paraphrase 9
Orlando, FL -- Shaquille O'Neal stroke for 37 points Friday night,
fueling the Orlando Magic to a 101 - 89 triumph over the Toronto Raptors.

> (progn (draft (dssE0)) (values))                              ;; Paraphrase 10

Orlando, FL -- Shaquille O'Neal fired in 37 points Friday night, pushing
the Orlando Magic to a 101 - 89 triumph over the Toronto Raptors.

> (progn (draft (dssE0)) (values))                              ;; Paraphrase 11
Orlando, FL -- Shaquille O'Neal pumped in 37 points Friday night,
lifting the Orlando Magic to a 101 - 89 triumph over the Toronto Raptors.

> (progn (draft (dssE0)) (values))                              ;; Paraphrase 12
Orlando, FL -- Shaquille O'Neal totaled 37 points Friday night while the
Orlando Magic coasted past the Toronto Raptors 101 - 89.

> (progn (draft (dssE0)) (values))                              ;; Paraphrase 13
Orlando, FL -- Shaquille O'Neal logged 37 points Friday night while the
Orlando Magic beat the Toronto Raptors 101 - 89.

>
```

Figure C.19: STREAK generating draft paraphrases

It also forces to link these two clauses by a temporal conjunction and the game result to be expressed using a *full* verb clause patterns. The differences between these three paraphrases is thus limited to the paradigmatic choices of open-class words fitting in this structure. For example, the verb expressing the main statistic ("*to rattle off*" in 1, "*to hit for*" in 4 and "*to strike for*" in 7), the conjunction linking the matrix and dependent clauses ("*as*" in 1 and 7 *vs.* "*while*" in 4) and the verb expressing the game result ("*to beat*" in 1, "*to rout*" in 4 and "*to triumph over*" in 7).

`form-flag2` used in paraphrase 2, 5 and 8 specifies the same top-level structure than `form-flag1` but forces the game result to be expressed using a *support* verb clause pattern. The paradigmatic variations within this pattern concern the choice of support verb ("*to post*" in 2, "*to clinch*" in 5 and "*to record*" in 8), the choice of head noun in the object NP of this support verb ("*triumph*" in 2 *vs.* "*victory*" in 5 and 8) and the choice of preposition introducing the losing team ("*against*" in 2 *vs.* "*over*" in 5 and 8).

`form-flag3` used in paraphrases 3, 6 and 9 forces STREAK to link the main statistic clause to the expression of the game result by a co-event clause. The paradigmatic variety in this case is the same than for `form-flag2` with additional choice of the head verb for the linking co-event clause ("*to rally*" in 3, "*to pace*" in 6 and "*to fuel*" in 9).

The paradigmatic variations in all paraphrases of Fig. C.2 result from random choices of lexicalization rules. In paraphrases 10 to 13, the syntagmatic variations, that were constrained by a form-flag in paraphrase 1 to 9, also result from random choices as well (of phrase planning rules) since the function `draft` is called without an `:sss` keyword parameter. In the first two of these randomly structured drafts a co-event clause is chosen to link the game result to the main statistic whereas in the last two a temporal conjunction is chosen.

C.2.2 Example run 4: Using revision to generate paraphrases

Having illustrated the two sources of paraphrasing power onto which STREAK can rely to vary its output during the drafting stage, I now turn to the four such sources onto which STREAK can rely during a revision increment: alternative revision rules, alternative draft constituents on which to perform the revision, and again alternative phrase planning rules and lexicalization rules for realizing the floating fact in the context of the revision.

Run 4, shown in Fig. C.20, illustrates the effects of these four sources. It contains a call to the function `revise1-para` which allows for parallel revision increments all incorporating the same additional fact to the same draft but each with a different revision rule. It thus take as input three parameters: a draft, an additional content unit and a list of revision rule names. `revise1-para` first prints its draft parameter. It then traverses its revision rule name list and repeatedly calls the function `revise1`[5]cf. sections C.1.2 for a description of this function.) to produce a different revision of the draft incorporating the additional content unit[6]. In *Run 4*, the draft is a basic sentence containing only the four fixed concepts. An example of semantic input for this concept combination was given in Fig. C.3. The main clause realizes the main statistic and a co-event dependent clause links this main statistic to the nominal realizing the game result. The additional content unit is of type losing streak extension. An example semantic input for this concept was given in Fig. C.4.

The first two elements in the revision rule name list `rules-E` are `Appositive Conjoin of Nominal`. The difference between *Draft1a* and *Draft1b* illustrates how STREAK can generate paraphrases by applying the same revision rule at different levels inside the draft structure. *Draft1a* results from the application of this apposition revision rule to the bottom level proper nominal "*the Toronto Raptors*" of *Draft0*. This nominal refers to the losing team whose streak was extended by the result of the reported game. It is embedded, as a qualifying PP complement, in another nominal conveying this game result "*a 101-89 triumph over the Toronto Raptors*". *Draft1b* results from the application of the same apposition revision rule but this time onto this embedding game result nominal as a whole. In these two examples, the phrase planner and lexicalizer building the linguistic form of the added constituent respectively produce *"losers of seven straight"* in *Draft1a*

[5] (

[6] *revise1-para* works by side-effects and always returns `nil`.

```
> (revise1-para (draft (dssE0) :sss (form-flag3)) (adssE1) (rules-E))

Draft 0:
Orlando, FL -- Shaquille O'Neal hit for 37 points Friday night, powering the Orlando
Magic to a 101 - 89 triumph over the Toronto Raptors.

Draft 1a:
Orlando, FL -- Shaquille O'Neal hit for 37 points Friday night, powering the Orlando
Magic to a 101 - 89 triumph over the Toronto Raptors, losers of seven straight.

Draft 1b:
Orlando, FL -- Shaquille O'Neal hit for 37 points Friday night, powering  the Orlando
Magic to a 101 - 89 triumph over the Raptors, Toronto's seventh setback in a row.

Draft 1c:
Orlando, FL -- Shaquille O'Neal hit for 37 points Friday night, powering the Orlando
Magic to a 101 - 89 triumph over the Toronto Raptors who lost their seventh straight.

Draft 1d:
Orlando, FL -- Shaquille O'Neal hit for 37 points Friday night, powering the Orlando
Magic to a 101 - 89 triumph over Toronto which sent the Raptors to their seventh loss
in a row.

Draft 1e:
Orlando, FL -- Shaquille O'Neal hit for 37 points Friday night, powering the Orlando
Magic to a 101 - 89 triumph over Toronto, handing the Raptors their seventh defeat in a
row.

Draft 1f:
Orlando, FL -- Shaquille O'Neal hit for 37 points Friday night, powering the Orlando
Magic to a 101 - 89 triumph over Toronto and riding the Raptors' losing streak to
seven.

NIL
>
```

Figure C.20: STREAK generating paraphrases by using a variety of revision rules

and *"Toronto's seventh setback in a row"*. These two forms correspond to the application of different phrase planning rules. For example whereas in the first the streak semantic element is mapped onto a pre-modifying adjective (*"straight"*), in the second it is mapped onto a synonymous post-modifying PP (*"in a row"*).

The subsequent two elements in the revision rule name list are `Adjoin of Relative Clause to Nominal`. This rule is applied to the same two constituents onto which `Appositive Conjoin of Nominal` where previously applied, respectively resulting in *Draft1c* and *Draft1d*. Again the difference in draft constituents onto which the rule is applied results in different linguistic forms for the new constituent attached by the revision. For example, in the relative clause *"who <u>lost</u> their seventh straight"* adjoined to the embedded proper nominal *"the Toronto Raptors"*, the losing nature of the streak is expressed by a verb (underlined) whereas in the relative clause *"which sent the Raptors to their seventh <u>loss</u>"* adjoined to the embedding common nominal *"a 101-89 triumph over the Toronto Raptors"* it is expressed by a noun (underlined).

The next element in the revision rule name list `rules-E` is `Adjoin of Non-finite Clause to Nominal`.

It differs from the previous rule only in terms of the type of clause adjoined by the revision. *Draft1e* results from the application of this rule to the same common nominal constituent onto which both `Appositive Conjoin of Nominal` and `Adjoin of Relative Clause to Nominal` were applied to respectively generate *Draft1b* and *Draft1c*. The contrast between *Draft1b*, *Draft1d* and *Draft1e* highlights the paraphrasing power provided by having a variety of revision rules applicable for the same purpose in the same context. The contrast between *"loss"* in the supplemental relative clause *"which sent the Raptors to their seventh loss in a row"* of *Draft1d* and *"defeat"* in the supplemental non-finite clause *"handing the Raptors their seventh defeat in a row"* in *Draft1e* illustrates how random choices of lexicalization rules also add to STREAK paraphrasing power during revision.

The next element in the revision rule name list `rules-E` is `Coordinative Conjoin of Clauses`. This rule is applied to a higher level draft constituent than in any of the previous revision increments of this example run: the whole co-event clause linking the game result to the main statistic. This constituent is four level higher in the draft syntactic three than the constituent onto which the revision rule was applied than to generate *Draft1a* and *Draft1c*.

C.2.3 Example run 5: Applying the same revision rules in different contexts

The last example run, shown in Fig. C.19, provides another case of incremental complex sentence generation. As opposed to the first case (*Run 1* presented in Section C.1.1) where a different revision rule was applied at each increment, two revision rules are applied twice each during the run in different textual contexts provided by the evolving draft. During the revision from *Draft 0* to *Draft 1* the rule `Coordinative Conjoin of Clause` is applied to the top-level clause of the draft expressing the main statistic. This revision adds a new player statistic. After the revision, the dependent clauses that were subordinated to this main statistic clause become subordinate to the whole new conjunction "*[Karl Malone provided 28 points] and [John Stockton added 27]*". During the revision from *Draft 2* to *Draft3* the same revision rule is applied again to add another statistic by the same player. This time the application is local to the second element of the conjunction built by the first application of that rule. It yields the embedded conjunction "*[[John Stockton added 27] and [∅ had 23 assists]]*". Similarly, the revision rule `Adjoin of Classifier to Nominal` is applied twice to add an historical fact of type record. It is first applied during the revision from *Draft 1* to *Draft 2* to the NP "*27*" summarizing Stockton's scoring performance and then during *Draft 3* to *Draft 4* to the NP *"23 assist"* summarizing Stockton's passing performance. The last revision in this run adds a historical fact of type winning streak extension. It uses the revision rule `Appositive Conjoin of Nominal`.

```
> (revise (draft (dssA0) :sss (form-flag3)) (float-stack-A) :verbose T)

Draft 0:
Los Angeles -- Karl Malone provided 28 points Saturday, fueling the Utah Jazz to a 127
- 111 win against the Los Angeles Clippers.

Draft 1 (lex-num = 29  depth = 7):
Los Angeles -- Karl Malone provided 28 points and John Stockton added 27 Saturday,
fueling the Utah Jazz to a 127 - 111 win against the Los Angeles Clippers.

Draft 2 (lex-num = 32  depth = 7):
Los Angeles -- Karl Malone provided 28 points and John Stockton added a season record
27 Saturday, fueling the Utah Jazz to a 127 - 111 win against the Los Angeles Clippers.

Draft 3 (lex-num = 36  depth = 7):
Los Angeles -- Karl Malone provided 28 points and John Stockton added a season record
27 and had 23 assists Saturday, fueling the Utah Jazz to a 127 - 111 win against the
Los Angeles Clippers.

Draft 4 (lex-num = 39  depth = 7):
Los Angeles -- Karl Malone provided 28 points and John Stockton added a season record
27 and had a league high 23 assists Saturday, fueling the Utah Jazz to a 127 - 111 win
against the Los Angeles Clippers.

Draft 5 (lex-num = 43  depth = 8):
Los Angeles -- Karl Malone provided 28 points and John Stockton added a season record
27 and had a league high 23 assists Saturday, fueling the Utah Jazz to their fourth
straight win, a 127 - 111 win against the Los Angeles Clippers.

((SSS ((DSS .... ))))
>
```

Figure C.21: Complex sentence generation where the same revision rule is used at different draft levels

Appendix D

Partially automating corpus analysis: the CREP software tool

A pervasive subtask of corpus analysis for generation knowledge acquisition is to search the corpus for occurrences of specific lexical items and/or syntactic forms. These items and forms can be searched for their own sake or as marks of a semantic message class. Such a search can be done by hand or by writing a scanning program specific to the search. After experimenting with both methods, I felt the need for a software tool automatically producing (and running) a scanning program from a flexible and high-level specification of the items to search for.

To address this need, I initiated and supervised the development of CREP a system that retrieves in a corpus all the sentences containing a lexico-syntactic pattern specified as a regular expression of words and/or part-of-speech tags. CREP was implemented by Duford. It is written on top of FLEX and currently uses Church's statistical tagger as the default part-of-speech tagger.

In this appendix, I give an overview of CREP and its use in assisting corpora analysis. I first briefly survey the syntax of CREP. I then discuss in detail three examples of corpus data analysis using CREP. The first example illustrates usage of CREP during the knowledge acquisition phase of the development of a generation system. It shows how to use CREP for the acquisition of lexical entries for a generator. The two other examples usage of CREP during the evaluation phase of the development of a generation system. The second example shows how to use CREP to search a test corpus for occurrences of a given realization pattern (*i.e.,* a given linguistic expression of a given domain concept combination). The process of encoding a realization pattern as a CREP expression is detailed on this second example. The third example shows how to use CREP to assess the portability of the knowledge structures used by a generator, in the case at hand the revision rules of STREAK. The process of encoding the signature of a revision rule[1] as a pair of CREP expression is detailed in this third example.

D.1 A brief overview of CREP syntax

In a CREP expression, the following operators can be used over words, over tags and recursively over CREP expressions:

- $exp1$ **;** $exp2$ specifying simple co-occurrence of $exp1$ and $exp2$
- $exp1$ **.** $exp2$ constraining $exp1$ to appear before $exp2$
- $exp1$ **N-** $exp2$ constraining $exp1$ to appear at a minimum distance of N words before $exp2$
- $exp1$ **N+** $exp2$ constraining $exp1$ to appear at a maximum distance of N words before $exp2$

[1] See Section 5.3.2.2 for the definition of *the signature* of a revision rule.

- $exp1$ **N=** $exp2$ constraining $exp1$ to appear at the exact distance of N words before $exp2$
- $exp1 \mid exp2$ specifying the occurrence of either $exp1$ or $exp2$
- $exp0$ **?** specifying the optional presence of $exp0$

A CREP expression can also contain:

- The $@BEG@$ keyword specifying the beginning of sentence position.
- The $@END@$ keyword specifying the end of sentence position.
- The **@@@** escape operator indicating that the expression surrounded by this operator should be interpreted in FLEX syntax instead of CREP syntax (this escaping operator makes all the FLEX operators usable in CREP expressions).

In order to modularly develop CREP expressions and re-use them in many searches, they can be put in a *definition file* that CREP accepts as input in addition to the main expression and the corpus to search. The other facilities of CREP include:

- An option to output the specific substring that matched the input expression in addition to the whole sentence containing it.
- An option to filter sentences matching different expressions in different files all at once. This option also allows simulation of 'at most one', 'exactly one' and 'zero' (*i.e.*,, negation) semantics with respect to the number of input expression matches inside a sentence (the default semantics being 'at least one').
- An option to overwrite the built in sentence delimiter using declarative rules.

See [Duford 1993] for a detailed presentation of these operators and options with many examples.

D.2 Using CREP for lexical knowledge acquisition

Its flexibility and user-friendliness make CREP a very useful tool at any point during the development of a corpus-based language generation application. In the present work, CREP proved useful beyond the *systematic* type of searches involved in the evaluation phase and exemplified in the next section. I also used it for the *opportunistic* type of searches needed during the system implementation phase.

For example, the following situation arose during the implementation of STREAK's surface mapping rules. One of these rules specifies what verbs can be used to express the rebounding statistic of a player. At the moment of writing this rule I remembered three such verbs:

- *"to grab"*, specific to rebounding statistics,
- *"to have"*, general for all types of statistics,
- *"to add"*, general for all types of statistics, but invalid as the opening statistic of the report

In order to widen the lexical coverage of STREAK, I searched for alternative verbs lexicalizing a rebounding statistics in the corpus. The CREP command to perform this search was:
`cat ../base/* | crep -w -e 'PLAYER 4- REB' -d ../defexpr/def1`
which instructs CREP to retrieve all the corpus sentences where an expression referring to a player is followed at a distance of at least four words by the expression of a word used to refer to rebounds. The detailed meaning of the options and parameters inside this command is the following:

- ../base is the path of the directory containing the corpus files
- -w is the CREP option to output the matching expression together sentence containing it

BOSTON (UPI) – Kevin Gamble scored 28 points Monday night and Joe Kleine added a career-high 20 rebounds , sending the Boston Celtics to a 110-89 victory over the Sacramento Kings for their eighth straight triumph.

Joe@NPNP Kleine@NPNP **added**@VBD a@AT career-high@JJ 20@CD rebounds@

MIAMI (UPI) – Steve Smith scored 30 points and Rony Seikaly grabbed a franchise-record 34 rebounds Wednesday night to lead the Miami Heat to their season-high fourth straight win , 125-106 over the Washington Bullets.

Rony@NPNP Seikaly@NPNP **grabbed**@VBD a@AT franchise-record@JJ 34@CD rebounds@

AUBURN HILLS , Mich. (UPI) – Joe Dumars scored 28 points and Dennis Rodman hauled in 21 rebounds Friday night to lead the Detroit Pistons to their fifth straight victory , a 107-103 win over the Cleveland Cavaliers.

Dennis@NPNP Rodman@NPNP **hauled**@VBN **in**@IN 21@CD rebounds@

ATLANTA (UPI) – Dominique Wilkins scored 30 points and Kevin Willis pulled down 16 rebounds to send the Atlanta Hawks to a 97-95 victory over the Los Angeles Clippers Saturday night.

Kevin@NPNP Willis@NPNP **pulled**@VBD **down**@IN 16@CD rebounds@

DENVER (UPI) – Rookies Mark Macon scored 18 points and Dikembe Mutombo ripped down 18 rebounds Thursday night , leading the Denver Nuggets to an 88-77 victory over the Minnesota Timberwolves.

Dikembe@NPNP Mutombo@NPNP **ripped**@VBD **down**@IN 18@CD rebounds@

Figure D.1: CREP search result for rebounding statistic verbs

- -e is the CREP option introducing the main expression parameter
- -d is the CREP option introducing the definition file parameter
- ../defexpr/def1 is the path of the definition file (which must contain definitions for both PLAYER and REB)

The entries for PLAYER and REB in the definition files were:

```
PLAYER  $@@@([A-Z][a-zA-Z]+(@NNS|@NP)|[^ @]+@NPNP[ ][^ @]+@NPNP)@@@$
REB     $(rebounds|boards)@$
```

The entry for player is surrounded by the '@@@' escape operator to indicate that it is written in FLEX syntax. It roughly means: "a reference to a player is defined as either a word starting with a capital letter and tagged as a proper noun[2] or as two consecutive such words. The definition for REB lists the two synonym units used for rebounding performances.

Part of the output of this search is given in fig. D.1. The first lines of each match contains the whole corpus sentence retrieved and the last line the particular phrase that caused the match. This output shows that three additional verbs to express rebounding statistics were uncovered: *"to pull down", "to haul in"* and *"to rip down"*. These verbs could then be included in the surface mapping rule of STREAK for the lexicalization of rebounding statistics.

D.3 Encoding realization patterns as CREP expressions

The first step towards computing the proportion of clusters in the first test corpus corresponding to usage of realization patterns abstracted from the acquisition corpus was to encode each realization pattern as a

[2] Or a noun phrase to match mistagged proper nouns.

CREP expression.

Recall from Section 2.4.2, that a realization pattern captures the mapping from a semantic unit combination onto a particular syntactic structure. It specifies the syntactic category used to express each semantic unit and the structural dependencies between these categories. For example, the realization pattern $R1$ in fig. D.2 specifies that a game result is expressed by a clause and a streak by a PP. It also specifies that the game result clause is the head constituent and the streak PP is attached to it as an adjunct. This is in contrast with pattern $R2$ where it is the streak that is expressed by the head clause and the game result by an adjunct PP.

Realization patterns abstract away from domain references, specific lexical items and low-level syntactic variations. Contrast for example the two corpus phrases given for each pattern in fig. D.2. Encoding a realization pattern as a CREP expression therefore involves two steps:

1. Write a syntagmatic pattern with most slots filled with a CREP sub-expression.
2. List the paradigmatic synonyms for each sub-expression in the definition file.

Step (1) is straightforward. It involves following the realization pattern from left to right and adjusting the distance operator parameters to account for low-level syntactic variations. For example, the main CREP expression $E2$ for the realization pattern $R2$ is given in figure D.3 with two matching corpus sentences aligned below it. To follow this example, note that CREP syntax relies on the following conventions:

- Uppercase distinguish sub-expressions from literals
- The '@' sign separates a word from its part-of-speech tag
- A blank space separates a word's part-of-speech tag from the next word

Step (2) requires more work. The definitions for the sub-expressions appearing $E2$ are given in fig D.4. Writing such sub-expression definitions involves identifying words and/or phrases satisfying a combination of semantic and syntactic constraints. For example the sub-expression `V_EXTEND` stands for verb (syntactic constraint) expressing a temporal extension (semantic constraint). It covers five verbs, *"to extend"*, *"to prolong"*, *"to stretch"*, *"to ride"* and *"to increase"*, which are synonyms for this particular sense.

To avoid missing clusters corresponding to a realization pattern requires exhaustive coverage of the vocabulary in the definition file. Such exhaustiveness can only be attained incrementally, starting from a manually defined bootstrapping vocabulary subset and then repeating corpus searches with CREP sub-expressions containing wild-cards. For example, the only nouns initially known for a win were *"win"* and *"victory"*. A corpus search with the expression:
$E3 =$ `a(n)?@ 0= SCORE 1= OVER 0= TEAM`
a sub-expression of $E2$ where no constraint is specified for the word appearing between `SCORE` and `OVER`, returned sentences like the two below (where the phrases matching the expression are in italics): "Indianapolis (UPI) – Rik Smits and Detlef Schrempf scored 23 points apiece Sunday , allowing the Indiana Pacers to extend their franchise - record home winning streak to 10 with *a 108-100* **triumph** *over the Philadelphia 76ers."*
or
"Los Angeles (UPI) – Michael Jordan scored 23 points and grabbed 10 rebounds and Scottie Pippen added 20 points Tuesday night to help the Chicago Bulls extend their winning streak to 10 games with *a 116-79* **rout** *of the Los Angeles Clippers."*.

Such sentences, allowed completing the definition of `N_WIN` with the words *"triumph", "rout", "drubbing", "blowing out", "decision", "romp", "upset", "humiliation", "trouncing"*, and *"defeat"*. These nouns are not all complete synonyms since some of them are appropriate for differently restricted score ranges. However, they all fill the same slot in E_2[3].

[3]Note that *"defeat"* which a priori expresses a loss, expresses a win in the particular context of this realization pattern where *"a SCORE defeat of TEAM"* can substitute for *"a SCORE win over TEAM"*.

Realization pattern $R1$ for the content unit combination:
<game-result(winner,loser,score),**streak(winner,aspect,result-type,length)**>

winner	game-result	loser	score			**length**	**streak+aspect**	**type**
agent	process	affected	score	result				
arg	head	arg	adjunct	adjunct				
proper	verb	proper	number	PP				
				prep	[det	ordinal	adj	noun]
Chicago	beat	Phoenix	99-91	**for**	**its**	**3rd**	**straight**	**win**
New York	defeated	Seattle	101-91	**for**	**its**	**4th**	**consecutive**	**victory**

Realization pattern $R2$ for the same combination:

winner	**aspect**		**type**	**streak**	**length**			score	game-result	loser
agent	process	affected/located			location	means				
arg	head	arg			adjunct	adjunct				
proper	verb	NP			PP	PP				
		det	participle	noun		prep	[det	number	noun	PP]
Utah	**extended**	**its**	**win**	**streak**	**to 6 games**	**with**	a	99-84	triumph	over Denver
Boston	**stretching**	**its**	**winning**	**spree**	**to 9 outings**	**with**	a	118-94	rout	of Utah

Figure D.2: Realization pattern examples (reproduced from Section 2.4.2)

TEAM	0=	V_EXTEND	0=	DET	0=	W_STREAK	0=	to@	0=	CARD	0=	GAMES	0=	...
Utah		extended		its		win streak		to		6		games		...
Boston		stretched		its		winning spree		to		7		outings		...

...	with	0=	a(n)?@	0=	SCORE	0=	N_WIN	0=	OVER	0=	TEAM
...	with		a		99-84		triumph		over		Denver
...	with		a		99-84		rout		of		Utah

Figure D.3: CREP expression E2 for realization pattern R2

```
;; Reference to a team = name of a city, or name of a franchise or both
TEAM            CITY|(the@ 1- (CITY 0=)? FRANCHISE)

CITY            (Boston|(New@ 0= York)|(New@ 0= Jersey)|Washington|Philadelphia|
                Charlotte|Orlando|Miami|Atlanta|Indiana|Cleveland|Detroit|Chicago|
                Milwaukee|Minnesota|Denver|(San@ 0= Antonio)|Dallas|Houston|Phoenix|
                Utah|(Los@ 0= Angeles)|(Golden@ 0= State)|Portland|Seattle|Sacramento)@

FRANCHISE       (76ers|Celtics|Knicks|Nets|Bullets|Sixers|Hornets|Magic|Heat|Hawks|
                Pacers|Cavaliers|Pistons|Bulls|Bucks|Timberwolves|Nuggets|Spurs|
                Mavericks|Rockets|Suns|Jazz|Lakers|Clippers|Warriors|
                ((Trail@ 0=)? Blazers)|Supersonics|(Super@ 0= Sonics)|Sonics|Kings)@

;; Verbs of temporal extension
V_EXTEND        (EXTEND|PROLONG|STRETCH|RIDE|INCREASE)
EXTEND          ((to@ 0=)? extend@)|extending@|extended@
PROLONG         ((to@ 0=)? prolong@)|prolonging@|prolonged@
STRETCH         ((to@ 0=)? stretch@)|stretching@|stretched@
RIDE            ((to@ 0=)? ride@)|riding@|rode@
INCREASE        ((to@ 0=)? increase@)|increasing@|increased@

;; Determiners = articles or possessive pronouns
DET             ARTICLE|POSS
ARTICLE         a(n)?@|the@
POSS            (my|your|her|his|its|our|their)@ (0= own@)?

;; Synonyms for ‘win streak’
WIN_STREAK        win(ning)?@ 0= (streak|spree|flurry|series)@

;; Cardinal numbers
CARD            @@@(one|two|three|four|five|six|seven|eight|nine|ten|eleven|[0-9]+)@@@@

;; Synonyms for ‘game’
GAMES          (game|decision|outing|contest)s@

;; Final score of a game
SCORE           (COMMA 0=)? ((CARD 1- CARD)|(@@@[0-9]+-[0-9]+@CD@@@)|
                (@@@[0-9][0-9][0-9][0-9]+@CD@@@)) (0= COMMA)?

;; Nouns expressing a victory
N_WIN           (victory|win|blowout|(blow@ 0= out)|defeat|rout|drubbing|triumph|
                decision|romp|upset)@

;; Preposition introducing losing team in NPs expressing game results
OVER            (over|versus|against|of)@
```

Figure D.4: CREP sub-expression definitions for realization pattern $R2$

To insure the completeness of the sub-expression definitions and the correctness of the main expressions, they were tested first on the acquisition corpus. It is only after these expressions yielded exactly the same results as those of the manual analysis on the acquisition corpus, that they were run on the first test corpus[4].

For such a systematic run, the CREP package includes a special shell taking as input a file where each expression E corresponding to a given realization pattern, is paired with a file name F. For each $< E, F >$ pair, this shell redirects the test corpus sentences matching E into F. It also redirects the sentences matching none of the expressions into a no-match file. Manual analysis of the no-match file is then required to get the values of the evaluation parameters.

D.4 Porting the signature of a revision tool: a detailed example

I now illustrate the process of porting the CREP expression pairs forming the approximate signature of a revision tool for the case of `Adjoin of Co-Event Clause to Clause`. The original, acquisition domain signature of this tool is shown in Fig. D.5. This figure contains only the sub-expressions differing from those for the signature of `Adjoin of Relative Clause to Top NP` which were already given in Fig. D.4 of the previous section. The corresponding, test domain signature for the same tool is shown in Fig. D.6. Note that for monotonic revisions like this one, the target pattern needs not contain the entire revised phrase but only the sub-phrase added to the source pattern by the revision.

An acquisition corpus cluster for each of the two target patterns of the original signature is given below:

- For TARGET-B1, (A_1): "*Utah Jazz hold on to defeat the Los Angeles Lakers 98-94* **, extending their home winning streak to 13 games.**"
- For TARGET-B2, (A_2): "*the Los Angeles Clippers defeat the Knicks 101-91* **, snapping a 12-game losing streak at the hands of New York dating back to February 23 , 1986.**"

There are three main differences between these two target patterns:

- Streak length expression (as a clause-modifying PP in TARGET-B1 *vs.* as an NP pre-modifier in TARGET-B2),
- Streak update type (extension in TARGET-B1 *vs.* interruption in TARGET-B2).
- Streak result type (winning in TARGET-B1 *vs.* losing in TARGET-B2).

Since these differences are independent, these two patterns can be merged in the more general[5]:

```
TARGET-C1 = (VG-EXTEND|VG-END) 0= DET 0= (STREAK-LENGTH 0=)? (WIN-STREAK|LOSE-STREAK)
```

This first generalization is motivated by the desire to factor out acquisition corpus artifacts during the process of porting the signature to the test corpus (and thus enhance the chances of finding a matching test sentence). Note that TARGET-B0 also covers streaks of unspecified length (since both alternative expressions for streak length are left optional), which is also desirable: although the length was always specified in the acquisition domain, it may not be the case in the test domain. On the test domain corpus, TARGET-C1 matched the following sentences:

1. (T_1): "The blue-chip Hang Seng Index , which sank 207.95 points Friday , soared 181.02 points to 9,177.95 **, snapping its three session losing streak.**"
2. (T_2): "In Australia , stock prices rallied in active trading on the Sydney Stock Exchange **snapping a two day decline.**

[4] Some result discrepancies between the manual analysis and the CREP expression runs on the acquisition corpus uncovered errors in the former.

[5] This more general pattern is glossed in the comments of Fig. D.6.

Original Signature = (< source-B1,target-B1 >), (< source-B1,target-B2 >)

```
;; Acquisition domain source pattern: team reference followed by a verb conveying
;; a positive outcome for that team, another team reference and a score
SOURCE_B1   TEAM 0= V_WIN 0= TEAM 0= SCORE

;; 1st acquisition domain target pattern: expresses winning streak extension
TARGET-B1   VG_EXTEND 0= DET 0= WIN_STREAK 0= @IN 0= STREAK_LENGTH

;; 1st acquisition domain target pattern: expresses losing streak interruption
TARGET-B2   VG_END 0= DET 0= STREAK_LENGTH 0= LOSE_STREAK

;; Extension verbs
VG_EXTEND   (extending|stretching|prolonging|riding)@

;; Interruption verbs
VG_END      (snapping|ending|breaking|stopping|halting|interrupting)@

;; Verbs conveying a positive outcome for the team reference preceding them
V_WIN       (((held@|hold@) 0= on@ 0= VI_WIN)|VP_WIN)
VI_WIN      to@TO 0= (defeat|beat|down|edge|pound|route|(cruise@ 0= past@)|...
VP_WIN      (defeated|beat|downed|edged|pounded|routed|(cruised@ 0= past@)|...

;; Nominals conveying streaks
WIN_STREAK    win(ning)?@ 0= (streak|spree|flurry|series)@
LOSE_STREAK   (slide@|(losing@ 0= streak@)|((losing@ 0=)? skid@)|drought@|slump@)

;; Constructs expressing the length of a streak
STREAK_LENGTH (CARD_PREMOD 1- GAME)

;; Cardinal number in pre-modifying position
CARD_PREMOD   @@@(two|three|four|five|six|seven|eight|nine|ten|eleven|[0-9]+)@@@@

;; Synonyms for game
GAME          (game|decision|outing|contest|session|day)@
```

Figure D.5: Original signature of `Adjoin of Co-Event Clause to Clause` in sports domain

Ported Signature = (< source-C1,target-C1 >)

```
;; Ported source pattern: a financial indicator reference, followed by a verb used in
;; the stock market domain to express a positive result and a score quantifying that
;; result
SOURCE_C1   INDICATOR 0= SM_V_WIN 0= SM_SCORE

;; Ported 1st target pattern: an -ing verb conveying an extension or interruption,
;; followed by a determiner, possibly a streak length and a noun conveying a streak
TARGET-C1   (VG_EXTEND|VG_END) 0= DET 0= (STREAK_LENGTH 0=)? (WIN_STREAK|LOSE_STREAK)

;; Stock market score
SM_SCORE    SM_CARD (0= SM_UNIT (1- SM_CARD)?)?
SM_UNIT     percent@|points@|shares@
SM_CARD     ((@@@([0-9]+,)?[0-9]+(\.[0-9]+)?@@@@) (0= (million@|billion@))?)|(CARD 0= COMMA 0= CARD)|CARD

;; Verbs expressing a positive outcome for the financial indicator preceding them
SM_W_WIN    (((held@|hold@) 0= on@ 0= SM_VI_WIN)|SM_VP_WIN)
SM_VI_WIN   (rising@|gaining@|jumping@|increasing@|climbing@|soaring@|...)
SM_VP_WIN   (rose@|gained@|jumped@|increased@|climbed@|soared@|...)

;; Financial indicator reference
INDICATOR   ((stock@ 0=)? prices@)|(the@ market@)|volume@|stocks@|INDEX
INDEX       ((T|t)he@) 0= (IC_PREMOD 0=)? IP_PREMOD 0= INDEX_HEAD 0= I_POSTMOD
IC_PREMOD   key@|@@@blue-chip@@@@|broader@|narrower@|@@@broader-based@@@@
IP_PREMOD   (Hang@ 0= Seng@)|Nikkei@|Tokyo@|Korean@|Singapore@|(Dow@ 0= Jones@)|...
IP_HEAD     ((I|i)ndex@)|((A|a)verage@)
IP_POSTMOD  of@ 0= CARD 0= (Industrials@|(Selected@ 0= Issues@))
```

Figure D.6: Signature of `Adjoin of Co-Event Clause to Clause` ported to the financial domain

3. (T_3): "The key All Ordinaries Index , which eased 7.9 points Wednesday , rose 9.6 points to 2,052.4 – **snapping its seven day losing streak.**

Searching for use of a tool in the test domain starts by considering each target pattern in turn, porting it to the test domain and analyzing the test sentences it matches. Presence in the test corpus of a sentence matching the ported *target* pattern constitutes in itself reasonable evidence for the usage of the revision tool. For example, sentences T_{1-3} above constitute reasonable evidence for the usage of `Adjoin of Clause as Co-event in Clause` in the stock market domain.

This evidence is confirmed by the co-presence in the test corpus of a surface decrement for any sentence matched by the ported target pattern. Finding such surface decrement also requires porting the *source* pattern(s) paired with the matching target pattern in the revision signature. This porting process can be guided by comparing the test corpus phrases matching the ported target signature to phrases that parallel them in the acquisition domain. In the case at hand, this means porting the source pattern SOURCE-B1 of Fig. 5.5 by comparing sentences T_1 and T_3 above, - which share the same source syntactic structure - to corresponding acquisition domain phrases such as

A_2: "*helping the Los Angeles Clippers defeat the Knicks 101-91* **, snapping a 12-game losing streak at the hands of New York dating back to February 23 , 1986.**"

This source pattern matches expressions of the `game-result` concept of the acquisition domain. However, there are conceptual discrepancies between `game-result` and its test domain counterpart, `variation`. First, `variation` is missing an affected role and the second TEAM sub-expression of SOURCE-B1 needs to be suppressed. Second, the agent role of `variation` is filled by a financial indicator instead of a basketball team. So the first TEAM sub-expression of SOURCE-B1 which matches basketball team names needs to be replaced by an INDICATOR sub-expression, matching financial indicators names. Defining this new sub-expression requires acquiring indicator names, which tend to be fairly complex (*e.g.,* "the Dow Jones average of 30 industrials"). See the sub-expressions below INDICATOR in Fig. D.6 for their encoding. After these conceptual discrepancies have been accounted for, the partially ported source pattern has become:

```
SOURCE-B2 = INDICATOR 0= V-WIN 0= SCORE.
```

As is, this expression cannot yet match any stock market corpus sentence. First, due to the suppression of the affected role in the test domain, different verbs express the positive result in each domain: transitive verbs (*e.g.,* "*The Gold Index soared*") instead of intransitive ones (*e.g.,* "*the Los Angeles Clippers defeated the New York Knicks*"). The sub-expression V-WIN covering the transitive verbs of the acquisition domain must thus be replaced by a sub-expression SM-V-WIN covering the intransitive ones of the test domain. Second, the quantities and units in basketball games scores (*e.g.,* "*101-91*") differ from those in financial indicator scores (*e.g.,* "*181.02 points to 9,177.95*"). The sub-expression SCORE in SOURCE-B2 covering basketball scores thus needs to be replaced by a sub-expression SM-SCORE covering stock market scores.

Finally, in test corpus, a historical fact[6] may be attached as relative clause to the NP referring to the indicator whose score is reported. For cases like these, where the only knowledge of the phenomenon comes from the few sentences that matched the ported target pattern, adjusting the source pattern can only be carried out incrementally using trial-and-error until a satisfying match is found. Since in both sentences T_1 and T_3 above, the additional relative clause (underlined) was seven word long[7], the first ported source expression tried was one allowing up to 10 words between the subject name and the verb:

```
SOURCE-C1 = INDICATOR 0= SM-V-WIN 10- SM-SCORE.
```

This expression matched the source test domain sentence:
"*The key Straits Times Industrials Index , which plummeted 65.45 points Thursday , soared 108.00 points to 2,302.86.*"

[6] Not belonging to the record or streak classes of historical facts studied in this thesis.
[7] Counting punctuation marks.

Since it matches some test corpus sentences, the pair < SOURCE-C1 , TARGET-C1 > constitutes the final approximate ported signature of the revision tool `Adjoin of Co-Event Clause to Clause`. The goal of this stock market domain analysis was *not* to discover new realization patterns. Instead it was only to evaluate the portability of the revison tools already discovered in the basketball domain. Therefore, for a given revision tool, once one < source , target > pair of realization patterns has been successfully ported, there is no need to consider the other such pairs associated with this tool. One can instead proceed to the next revision tool.

Bibliography

[Abella 1994] A. Abella. From pictures to words: generating locative descriptions of objects in an image. In *Proceedings of the Image Understanding Workshop*, Monterrey, CA, November 1994.

[Anderson 1985] D. Anderson. *Contemporary sports reporting.* Nelson-Hall, Chicago, IL, 1985.

[Andre *et al.* 1993] E. Andre, W. Finkler, T. Graf, A. Rist, A. Schauder, and W. Wahlster. WIP, the automatic synthesis of multimedia presentation. In M. Maybury, editor, *Intelligent Multimedia Interfaces.* AAAI Press, Cambridge, MA, 1993.

[Appelt 1985] D. Appelt. *Planning Natural Language Utterances.* Studies in Natural Language Processing. Cambridge University Press, 1985.

[Bateman *et al.* 1990] J.A. Bateman, R.T. Kasper, J.D. Moore, and R.A. Whitney. A general organization of knowledge for natural language processing: the PENMAN Upper-Model. Technical report, ISI, Marina del Rey, CA, 1990.

[Borko 1975] H. Borko. *Abstracting Concepts and Methods.* Academic Press, New York, NY, 1975.

[Bourbeau *et al.* 1990] L. Bourbeau, D. Carcagno, E. Goldberg, R. Kittredge, and A. Polguere. Bilingual generation of weather forecasts in an operations environment. In *Proceedings of the 13th International Conference on Computational Linguistics*, Helsinki University, Finland, 1990. COLING.

[Bresnan and Kaplan 1982] J. Bresnan and R. Kaplan. Lexical Functional Grammars: a formal system for grammatical representation. In J. Bresnan, editor, *The mental representation of grammatical relations.* MIT Press, Cambridge, MA, 1982.

[Carcagno and Iordanskaja 1993] D. Carcagno and L. Iordanskaja. Content determination and text structuring: two interrelated processes. In H. Horacek and M. Zock, editors, *New Concepts in Natural Language Generation: Planning, Realization and Systems.* Frances Pinter, London and New York, 1993.

[Cline and Nutter 1991] B.E. Cline and J.T. Nutter. Conceptual revision for natural language generation, 1991. Student session of the 29th Annual Meeting of the ACL.

[Contant 1986] C. Contant. Generation automatique de texte: application au sous-language boursier, 1986. M.A. thesis, Universite de Montreal.

[Dale 1992] R. Dale. *Generating Referring Expressions.* ACL-MIT Press Series in Natural Language Processing, Cambridge, Ma., 1992.

[Danlos 1986] L. Danlos. *The linguistic basis of text generation.* Studies in Natural Language Processing. Cambridge University Press, 1986.

[Davey 1978] A. Davey. *Discourse production.* Edinburgh University Press, 1978.

[De Smedt and Kempen 1987] K. De Smedt and G. Kempen. Incremental sentence production, self-correction and coordination. In Gerard Kempen, editor, *Natural Language Generation: New Results in Artificial Intellligence, Psychology and Linguistics.* Martinus Ninjhoff Publishers, 1987.

[De Smedt 1990] K. De Smedt. IPF: an incremental parallel formulator. In R. Dale, C.S. Mellish, and M. Zock, editors, *Current Research in Natural Language Generation.* Academic Press, 1990.

[Duford 1993] D. Duford. CREP: a regular expression-matching textual corpus tool. Technical Report CUCS-005-93, Columbia University, 1993.

[Elhadad and Robin 1992] M. Elhadad and J. Robin. Controlling Content Realization with Functional Unification Grammars. In R. Dale, H. Hovy, D. Roesner, and O. Stock, editors, *Aspects of Automated Natural Language Generation*, pages 89–104. Springler Verlag, 1992.

[Elhadad 1990] M. Elhadad. Types in Functional Unification Grammars. In *Proceedings of the 28th Annual Meeting of the Association for Computational Linguistics*, Detroit, MI, 1990. ACL.

[Elhadad 1993a] M. Elhadad. FUF: The universal unifier - user manual, version 5.2. Technical Report CUCS-038-91, Columbia University, 1993.

[Elhadad 1993b] M. Elhadad. *Using argumentation to control lexical choice: a unification-based implementation.* PhD thesis, Computer Science Department, Columbia University, 1993.

[Fawcett 1987] R.P. Fawcett. The semantics of clause and verb for relational processes in English. In M.A.K. Halliday and R.P. Fawcett, editors, *New developments in systemic linguistics.* Frances Pinter, London and New York, 1987.

[Fensch 1988] T. Fensch. *The sports writing handbook.* Lawrence Erlbaum Associates, Hillsdale, NJ, 1988.

[Fries 1970] P. Fries. *Tagmeme sequences in the English noun phrase.* Number 36 in Summer institute of linguistics publications in linguistics and related fields. Benjamin F. Elson for The Church Press Inc., Glendale, CA, 1970.

[Gabriel 1988] R. Gabriel. Deliberate writing. In D.D. McDonald and Bolc L., editors, *Natural Language Generation Systems.* Springer-Verlag, New York, NY, 1988.

[Gopen and Swan 1990] G.D. Gopen and J.A. Swan. The science of scientific writing. *American Scientist*, 78:550–558, 1990.

[Gross 1984] M. Gross. Lexicon-Grammar and the syntactic analysis of French. In *Proceedings of the 10th International Conference on Computational Linguistics*, pages 275–282. COLING, 1984.

[Halliday 1985] M.A.K. Halliday. *An introduction to functional grammar.* Edward Arnold, London, 1985.

[Harbusch 1994] K. Harbusch. Towards an Integrated Generation Approach with Tree-Adjoining Grammars. *Computational Intelligence*, 10(4):579–590, 1994.

[Hatzivassiloglou and McKeown 1993] Vasileios Hatzivassiloglou and Kathleen McKeown. Towards the Automatic Identification of Adjectival Scales: Clustering Adjectives According to Meaning. In *Proceedings of the 31st Annual Meeting of the ACL*, pages 172–182, Columbus, Ohio, June 1993. Association for Computational Linguistics.

[Herskovits 1986] A. Herskovits. *Language and spatial cognition: an interdisciplinary study of the prepositions of English.* Studies in Natural Language Processing. Cambridge University Press, 1986.

[Hirschman and Cuomo 1994] L. Hirschman and D. Cuomo. Report from the ARPA Workshop on Evaluation of Human Computer Interfaces. Technical Report MP 94B0000259, Mitre, Bedford, Ma., 1994.

[Hovy 1988] E. Hovy. *Generating natural language under pragmatic constraints.* L. Erlbaum Associates, Hillsdale, N.J., 1988.

[Hovy 1990] E. Hovy. Unresolved issues in paragraph planning. In R. Dale, C.S. Mellish, and M. Zock, editors, *Current Research in Natural Language Generation.* Academic Press, 1990.

[Hovy 1991] E. Hovy. Approaches to the planning of coherent text. In C. Paris, W. Swartout, and Mann. W.C., editors, *Natutal Language Generation in Artificial Intelligence and Computational Linguistics.* Kluwer Academic Publishers, Boston, 1991.

[Inui *et al.* 1992] K. Inui, T. Tokunaga, and H. Tanaka. Text revision: a model and its implementation. In R. Dale, E. Hovy, D. Roesner, and O. Stock, editors, *Aspects of Automated Natural Language Generation*, pages 215–230. Springler-Verlag, 1992.

[Iordanskaja *et al.* 1994] L. Iordanskaja, M. Kim, R. Kittredge, B. Lavoie, and A. Polguere. Generation of extended bilingual statistical reports. In *Proceedings of the 15th International Conference on Computational Linguistics.* COLING, 1994.

[Jacobs 1985] P. Jacobs. PHRED: a generator for natural language interfaces. *Computational Linguistics*, 11(4):219–242, 1985.

[Joshi *et al.* 1975] A.K. Joshi, L.S. Levy, and M. Takahashi. Tree Adjunct Grammars. *Journal of Computer and System Sciences*, 10(1):136–163, 1975.

[Joshi 1987] A.K. Joshi. The relevance of Tree-Adjoining Grammar to generation. In Gerard Kempen, editor, *Natural Language Generation: New Results in Artificial Intellligence, Psychology and Linguistics.* Martinus Ninjhoff Publishers, 1987.

[Kalita 1989] J. Kalita. Automatically generating natural language reports. *International Journal of Man-Machine Studies*, (30):399–423, 1989.

[Kay 1979] M. Kay. Functional Grammar. In *Proceedings of the 5th Annual Meeting of the Berkeley Linguistic Society*, 1979.

[Kukich *et al.* 1994] K. Kukich, K. McKeown, J. Shaw, J. Robin, N. Morgan, and J. Phillips. User-needs analysis and design methodology for an automated document generator. In A. Zampolli, N. Calzolari, and M. Palmer, editors, *Current Issues in Computational Linguistics: In Honour of Don Walker.* Kluwer Academic Press, Boston, 1994.

[Kukich 1983] K. Kukich. *Knowledge-based report generation: a knowledge engineering approach to natural language report generation.* PhD thesis, University of Pittsburgh, 1983.

[Kukich 1985] K. Kukich. The feasibility of automatic natural language generation report. In *Proceedings of the 18th Hawaii international conference on system sciences*, 1985.

[Lenat and Guha 1989] D.B. Lenat and R.V Guha. *Building large knowledge base systems: representation and inference in the Cyc project.* Addison-Wesley, 1989.

[Levi 1978] J. Levi. *The syntax and semantics of complex nominals.* Academic Press, 1978.

[Levin 1993] B. Levin. *English verb classes and alternations: a preliminary investigation.* University of Chicago Press, 1993.

[Lyons 1977] J. Lyons. *Semantics.* Cambridge University Press, 1977.

[Mann and Matthiessen 1983] W.C. Mann and C.M. Matthiessen. NIGEL: a systemic grammar for text generation. Technical Report ISI/RR-83-105, ISI, Marina del Rey, CA, 1983.

[Mann and Moore 1981] W.C. Mann and J.A. Moore. Computer Generation of Multiparagraph English Text. *Computational Linguistics*, 7(1):17–29, 1981.

[Mann 1983] W.C. Mann. An overview of the PENMAN text generation system. Technical Report ISI/RR-83-114, ISI, Marina del Rey, CA, 1983.

[Maybury 1990] M.T. Maybury. Using discourse focus, temporal focus and spatial focus to generate multisentential text. In *Proceedings of the 5th International Workshop on Natural Language Generation*, Pittsburgh, PA, 1990.

[McCoy *et al.* 1992] K. McCoy, K. Vijay-Shanker, and G. Yang. A functional approach to generation with TAG. In *Proceedings of the 30th Annual Meeting of the Association for Computational Linguistics*. ACL, 1992.

[McDonald and Pustejovsky 1985] D. McDonald and J.D. Pustejovsky. TAGs as a formalism for generation. In *Proceedings of the 23rd Annual Meeting of the Association for Computational Linguistics*. ACL, 1985.

[McKeown and Swartout 1987] K.R. McKeown and W.R. Swartout. Language generation and explanation. *The Annual Review of Computer Science*, (2):401–449, 1987.

[McKeown *et al.* 1990] K. R. McKeown, M. Elhadad, Y. Fukumoto, J.G. Lim, C. Lombardi, J. Robin, and F.A. Smadja. Text generation in COMET. In R. Dale, C.S. Mellish, and M. Zock, editors, *Current Research in Natural Language Generation*. Academic Press, 1990.

[McKeown *et al.* 1993] K. R. McKeown, J. Robin, and M. Tanenblatt. Tailoring lexical choice to the user's vocabulary in multimedia explanation generation. In *Proceedings of the 31st Annual Meeting of the Association for Computational Linguistics*. ACL, 1993.

[McKeown *et al.* 1995] K. McKeown, J. Robin, and K. Kukich. Generating concise natural language summaries. *Information Processing and Management*, 1995. (To appear in "Special Issue on Summarization").

[McKeown 1985] K. R. McKeown. *Using Discourse Strategies and Focus Constraints to Generate Natural Language Text*. Studies in Natural Language Processing. Cambridge University Press, 1985.

[Mel'cuk and Pertsov 1987] I.A. Mel'cuk and N.V. Pertsov. *Surface-syntax of English, a formal model in the Meaning-Text Theory*. Benjamins, Amsterdam/Philadelphia, 1987.

[Mel'cuk and Polguere 1987] I.A. Mel'cuk and A. Polguere. A formal lexicon in the Meaning-Text Theory. *Computational Linguistics*, 13(3-4):261–275, 1987.

[Mencher 1984] M. Mencher. *News reporting and writing*. Wm. C. Brown Publishers, Dubuque, Iowa, 1984.

[Meteer *et al.* 1987] M.W. Meteer, D.D. McDonald, S.D. Anderson, D. Forster, L.S. Gay, A.K. Huettner, and P. Sibun. Mumble-86: Design and implementation. Technical Report COINS 87-87, University of Massachussets at Amherst, Ahmerst, Ma., 1987.

[Meteer 1990] M.W. Meteer. *The generation gap: the problem of expressibility in text planning*. PhD thesis, University of Massachussets at Ahmerst, 1990. Also available as BBN technical report No. 7347.

[Meteer 1991] M. Meteer. The implications of revisions for natural language generation. In C. Paris, W. Swartout, and W.C. Mann, editors, *Natutal Language Generation in Artificial Intelligence and Computational Linguistics*. Kluwer Academic Publishers, Boston, 1991.

[MUC-4 1992] Proceedings of the Fourth Message Understanding Conference. In *Proceedings of the Fourth Message Understanding Conference*. DARPA Software and Intelligent Systems Technology Office, 1992.

[Nogier 1990] J.F. Nogier. *Un systeme de production de language fonde sur le modele des graphes conceptuels*. PhD thesis, Universite de Paris VII, 1990.

[Paris 1987] C. L. Paris. *The use of explicit user models in text generation: tailoring to a user's level of expertise*. PhD thesis, Columbia University, 1987. Also available as technical report CUCS-309-87.

[Pavard 1985] B. Pavard. La conception de systemes de traitement de texte. *Intellectica*, 1(1):37–67, 1985.

[Polguere 1990] A. Polguere. *Structuration et mise en jeu procedurale d'un modele linguistique declaratif dans un cadre de generation de texte*. PhD thesis, Universite de Montreal, Quebec, Canada, 1990.

[Pollard and Sag 1987] C. Pollard and I.A. Sag. *Information-based Syntax and Semantics - Volume 1*, volume 13 of *CSLI Lecture Notes*. University of Chicago Press, Chicago, Il, 1987.

[Quirk *et al.* 1985] R. Quirk, S. Greenbaum, G. Leech, and J. Svartvik. *A comprehensive grammar of the English language.* Longman, 1985.

[Rau 1987] L.F. Rau. Information retrieval in never ending stories. In *Proceedings of the 6th National Conference on Artificial Intelligence*, pages 317–322, Washington, 1987.

[Reiter 1991] E.B. Reiter. A New Model for Lexical Choice for Open-Class Words. *Computational Intelligence*, (7), December 1991.

[Reiter 1994] E.B. Reiter. Has a consensus natural language generation architecture appeared and is it psycholinguistically plausible? In *Proceedings of the 7th International Workshop on Natural Language Generation*, pages 163–170, June 1994.

[Robin 1990] J. Robin. Lexical Choice in Natural Language Generation. Technical Report CUCS-040-90, Columbia University, 1990.

[Roesner 1987] D. Roesner. SEMTEX: a text generator for German. In Gerard Kempen, editor, *Natural Language Generation: New Results in Artificial Intellligence, Psychology and Linguistics.* Martinus Ninjhoff Publishers, 1987.

[Roth *et al.* 1991] S. Roth, J. Mattis, and X. Mesnard. Graphics and natural language as component of automatic explanation. In J.W. Sullivan and S.W. Tyler, editors, *Intelligent user-interfaces*, ACM press frontier. Addison-Wesley, 1991.

[Rubinoff 1992] R. Rubinoff. *Negotiation, feedback and perspective within natural language generation.* PhD thesis, Computer Science Department, University of Pennsylvania, 1992.

[Scott and Souza 1990] D.R. Scott and C.S. Souza. Getting the message across in RST-based text generation. In R. Dale, C.S. Mellish, and M. Zock, editors, *Current Research in Natural Language Generation.* Academic Press, 1990.

[Shabes *et al.* 1988] Y. Shabes, A. Abeille, and A.K. Joshi. Parsing strategies with 'lexicalized' grammars: application to tree adjoining grammars. In *Proceedings of the 12th International Conference on Computational Linguistics*, Karl Marx University of Economics, Budapest, 1988. COLING.

[Shieber and Shabes 1991] S.M. Shieber and Y. Shabes. Generation and synchronous tree-adhoining grammars. *Computational Intelligence*, 7(4):220–228, December 1991.

[Simonin 1985] N. Simonin. Essai de modelisation de l'expertise en redaction de textes. In *Proceedings of COGNITIVA 85.* COGNITIVA, 1985.

[Skaliar 1994] D. Skaliar. A FUF interpreter based on graph representations. Technical report, Ben Gurion University of the Negev, Beer Sheva, Israel, 1994.

[Smadja and McKeown 1991] F.A. Smadja and K.R. McKeown. Using collocations for language generation. *Computational Intelligence*, 7(4):229–239, December 1991.

[Smadja 1991] F. Smadja. *Retrieving Collocational Knowledge from Textual Corpora. An Application: Language Generation.* PhD thesis, Computer Science Department, Columbia University, 1991.

[Strzalkowski 1994] T. (Ed.) Strzalkowski. *Reversible grammars in natural language processing.* Kluwer Academic Publishers, Boston, 1994.

[Talmy 1976] L. Talmy. Semantic causative types. In M. Shibatani, editor, *The grammar of causative constructions*, volume 6 of *Syntax and semantics.* Academic Press, London, 1976.

[Talmy 1983] L. Talmy. How language structures space. In H.L. Pick and L.P. Acredolo, editors, *Spatial orientation: theory, research and application.* Plenum Press, New York/London, 1983.

[Talmy 1985] L. Talmy. Lexicalization patterns: semantic structure in lexical form. In T. Shopen, editor, *Grammatical categories and the lexicon*, volume 3 of *Language typology and syntactic description*. Cambridge University Press, 1985.

[Thompson and Longacre 1985] S.A. Thompson and R.E. Longacre. Adverbial clauses. In T. Shopen, editor, *Complex constructions*, volume 2 of *Language typology and syntactic description*. Cambridge University Press, 1985.

[Vaughan and McDonald 1986] M. Vaughan and D.D. McDonald. A Model of Revision in Natural Language Generation. In *Proceedings of the 23rd Annual Meeting of the Association for Computational Linguistics*, Columbia University, New York, 1986. ACL.

[Vendler 1968] Zeno Vendler. *Adjectives and Nominalizations*. Mouton, The Hague, Paris, 1968.

[Vijay-Shanker 1992] K. Vijay-Shanker. Using descriptions of trees in a tree adjoining grammar. *Computational Linguistics*, 18(4):483–517, 1992.

[Wahlster *et al.* 1993] W. Wahlster, E. Andre, W. Finkler, H.J. Profitlich, and T. Rist. Plan-based integration of natural language and graphics generation. *Artificial Intelligence*, (63):387–427, 1993.

[Winograd 1983] T. Winograd. *Language as a cognitive process*. Addison-Wesley, 1983.

[Wong and Simmons 1988] W.K.C. Wong and R.F Simmons. A blackboard model for text production with revision. In *Proceedings of the AAAI workshop on text-planning and realization*, St-Paul, MN, 1988. AAAI.

[Yang *et al.* 1991] G. Yang, K. McCoy, and K. Vijay-Shanker. From functional specification to syntactic structure: systemic grammar and tree adjoining grammars. *Computational Intelligence*, 7(4), December 1991.

[Yazdani 1987] M. Yazdani. Reviewing as a component of the text generation process. In Gerard Kempen, editor, *Natural Language Generation: New Results in Artificial Intellligence, Psychology and Linguistics*. Martinus Ninjhoff Publishers, 1987.

[Zock *et al.* 1992] M. Zock, M. Kay, D. Carcagno, J.F. Nogier, M. Nossin, K. DeSmedt, and F. Namer. Automatic text generation, a tool for the buisiness world? In *Proceedings of the Annual Expert Systems Conference*, Avignon, France, 1992.

[Zock 1988] M. Zock. Natural languages are flexible tools, that's what makes them hard to explain, to learn and to use. In M. Zock and G. Sabah, editors, *Advances in Natural Language Generation: an Interdisciplinary Perspective*. Pinter and Ablex, 1988.