Floating Constraints in Lexical Choice

Michael Elhadad[*]
Ben Gurion University in the Negev

Kathleen McKeown[†]
Columbia University

Jacques Robin[‡]
Universidade Federal de Pernambuco

Lexical choice is a computationally complex task, requiring a generation system to consider a potentially large number of mappings between concepts and words. Constraints that aid in determining which word is best come from a wide variety of sources, including syntax, semantics, pragmatics, the lexicon, and the underlying domain. Furthermore, in some situations, different constraints come into play early on, while in others, they apply much later. This makes it difficult to determine a systematic ordering in which to apply constraints. In this paper, we present a general approach to lexical choice which can handle multiple, interacting constraints. We focus on the problem of floating constraints, *semantic or pragmatic constraints which float, appearing at a variety of different syntactic ranks, often merged with other semantic constraints. This means that multiple content units can be realized by a single surface element, and conversely, that a single content unit can be realized by a variety of surface elements. Our approach uses the Functional Unification Formalism (*FUF*) to represent a generation lexicon, allowing for declarative and compositional representation of individual constraints.*

1. Introduction

Given a request to communicate, a language generator typically must select information from an underlying domain representation, and determine how to order this information, ultimately realizing the representation as sentences by selecting words, and linearly ordering them under the syntactic constraints of the language. The problem of determining what words to use for the concepts in the domain representation is termed *lexical choice.* In an effort to make domain representations independent of language, there may be a variety of different words that can be used to express any concept in the domain, and a language generator must choose which one is most appropriate in the current context. A one-to-one mapping between each domain concept and a word of the language would imply that concepts are represented by words, clearly an undesirable situation. Just as there is no reason to assume that a concept uniquely determines a word, there is no reason to assume that a single concept must map to a single word; a domain concept may be expressed by multiple words, or conversely, a single word may express a combination of concepts (Talmy, 1985) (Zock, 1988).

Avoiding encoding any assumptions about the mapping between domain and language has the benefit of portability; the architecture and some knowledge sources of the generator can be re-used for a variety of different applications in quite different domains. However, it means the task of lexical choice is computationally complex, requiring consideration of a potentially large number of mappings between concepts and words. This

* Mathematics and Computer Science Dept, Beer Sheva, 84105 Israel, elhadad@cs.bgu.ac.il
† Computer Science Dept, New York, NY 10027 USA, kathy@cs.columbia.edu
‡ Departamento de Informática, Recife, PE 50740-540 Brazil, jr@di.ufpe.br

is complicated by the fact that constraints on lexical choice come from a wide variety of sources:

- Syntax (the choice of a particular verb influences the syntactic forms that can be used to realize its arguments, which in turn constrains the words used to lexicalize these arguments). For example, if the main verb "to allow" is selected, then the object must be either a clause ("allow one to select") or a noun-group ("allow the selection"). [1]
- Semantics (the concept itself and how it is taxonomized in the domain influence which word should be used). For example, when discussing basketball, the words "rebound" and "point" realize distinct concepts under the generic concept of a player "performance".
- The lexicon (the choice of one word can constrain the choice of other words in a sentence). For example, the selection of "rebound" as object noun would entail preferring "to grab" over "to score" as main verb, while the selection of "point" would entail the opposite verb choice, since "to grab rebounds" and "to score points" are lexical collocations, whereas "? to score rebounds" and "? to grab points" are *not.*
- The domain (different words are used to refer to the same concept in different domain sub-languages). For example, "rebound" means different things in the basketball domain and in the stock-market domain ("IBM rebounded from a 3 day loss" vs. "Magic grabbed 20 rebounds").
- Pragmatics (information about speaker intent, hearer background, or previous discourse plays a role). This may lead to the decision to refer to the same situation as a glass "half full" or "half empty".

Furthermore, interaction between constraints is multi-directional, making it difficult to determine a systematic ordering in which constraints should be taken into account. In fact, earlier work on lexical choice (Danlos, 1986) implied that a new ordering of constraints, and thus a new architecture for lexical choice, must be developed for each new domain.

In this paper, we present a general approach to lexical choice which can handle multiple, interacting constraints. Our architecture positions the lexical choice module between a language generator's content planner and surface sentence generator, in order to take into account conceptual, pragmatic and linguistic constraints on word choice. We show how the Functional Unification Formalism (FUF) (Elhadad, 1993a), originally developed for representing syntactic grammars (Kay, 1979), can be used to represent a generation lexicon, allowing for declarative and compositional representation of independent constraints. Ordering on how constraints are applied is determined dynamically through unification, allowing for different orderings as required. Since any approach must deal with a combinatorial explosion of possible mappings and ordering of constraints, computational efficiency is in general an issue. We show control techniques we have developed within FUF to reduce overall search. In this paper, we illustrate our model for lexical choice as it has been implemented in ADVISOR-II (Elhadad, 1993c), a system that can advise students about course selection, but we also draw on examples from two other sys-

1 The options are different in French for example, where the corresponding verb governs a VP "permet de sélectioner".

tems based on the same model but within different generation architectures[2]: STREAK, a system for generating basketball game summaries (Robin, 1994a) (Robin and McKeown, 1996) and COOK (Smadja and McKeown, 1991), a system that generates stock market reports. We have used this same model for lexical choice in other systems we have developed such as COMET (McKeown et al., 1990), a multimedia explanation system for equipment maintenance and repair, and PLANDOC (Kukich et al., 1994), an automated documentation system under collaborative development with Bellcore.

We focus on the problem of *floating constraints*, constraints which cannot be mapped in a systematic way from an input conceptual representation to the linguistic structure. Instead, such constraints float, appearing at a variety of different levels in the resulting linguistic structure, depending on other constraints in the input. Such constraints pose problems (see discussion in (Elhadad and Robin, 1992)) for the top-down recursive building of the linguistic structure used by most generation algorithms (Meteer et al., 1987) (Shieber et al., 1990); these algorithms typically only handle *structural constraints*, constraints that are consistently expressed at a given linguistic rank (*e.g.,* the sentence, clause, group or word rank) (Halliday, 1985) in the application domain sub-language. We consider two different types of floating constraints:

1. *inter-lexical* constraints, which arise from restrictions on lexical co-occurrences such as collocations (Smadja, 1991) (they are orthogonal to the mapping from input content units onto output linguistic form since they both originate from the lexicon and act upon the lexicon)
2. *cross-ranking* constraints, which arise from the fact that an input network of content units is not isomorphic with the resulting linguistic structure, allowing a single content unit to be realized by surface elements of various linguistic ranks (cross-ranking proper), or multiple content units to be realized by the same surface element (merging).

Sentences (1) and (2) below, generated by COOK, illustrate cross-ranking constraints. They show how time and manner can be mapped to two different surface elements of different syntactic rank in the sentence, among many other possibilities. Sentences (3) and (4), generated by STREAK, show how game result and manner can be realized as two separate surface elements or can be merged into a single element, the verb.

1. "Wall Street Indexes *opened* **strongly**." (*time* in verb, **manner** as adverb)
2. "Stock indexes **surged** *at the start of the trading day*." (*time* as PP, **manner** in verb)
3. "The Denver Nuggets <u>beat</u> the Boston Celtics **with a narrow margin**, 102-101." (<u>game result</u> in verb, **manner** as PP).
4. "The Denver Nuggets **<u>edged out</u>** the Boston Celtics 102-101." (<u>game result</u> and **manner** in verb)

In these examples, the input conceptual constraints (time and manner) float, appearing at a variety of different syntactic ranks (here, verb and circumstantial), and are sometimes merged with other semantic constraints.

2 The differences between the system architectures of these three systems are discussed in Sect. 6.1.2.

Which content units are floating and which are structural depends on the domain and the particular target sub-language. Our corpus analysis of the basketball domain, for example, indicates that historical knowledge is floating, whereas game result information is structural. Similarly, in the student advising domain, we found course evaluation (*e.g.,* how difficult or interesting a course is) to be floating, whereas the description of the assignments required (*e.g.,* how many there are or whether they involve writing essays, software or proofs) in a course is structural.

Floating constraints have not been addressed in a general way in previous work; most systems implicitly hard-wire the choices or permit only one or two of many possibilities. In contrast, our model for lexical choice accommodates floating constraints, resulting in a system which has a high degree of paraphrasing power.

In the following sections, we first present our general model for lexical choice illustrating it with a relatively simple example. We then discuss different types of constraints and the problems they pose, presenting the techniques we have developed within FUF to address these issues turning from structural constraints, to pragmatic cross-ranking constraints, and to inter-lexical constraints. Finally, we compare our approach with other work on lexical choice, closing with a summary of our contributions.

2. An architecture for lexical choice

The place of lexical choice in the overall architecture of generation systems has varied from project to project. Due to the varied nature of the constraints on lexical choice, exactly how lexical choice is done often depends on which type of constraints a system design accounts for. For example, if syntactic and lexical constraints are the research focus, it may make sense to delay lexical choice until late in the generation process, during syntactic realization. If only conceptual constraints are accounted for, lexical choice may be done early on, for example during content planning by associating concepts with the words or phrases that can realize them.

In this section, we describe a general model for lexical choice as part of an overall generation system architecture. Due to the wide variety of constraints on word selection that we consider, lexicalization is positioned after the content of the generated text has been determined and before syntactic realization takes place. We detail the nature of input and output to the lexical choice module thus specifying the tasks the lexical choice module performs, and the tasks that are expected to be done elsewhere in the system. We illustrate, through a relatively simple example, which depends on a single type of constraint, how FUF and unification are used for lexicalization. Our criteria for a model for lexical choice are fourfold:

1. It must be able to use the full variety of constraints whether pragmatic, semantic, lexical or syntactic;
2. It must be able to apply constraints in a flexible order;
3. It must avoid encoding assumptions about the mapping between domain concepts and lexical structure;
4. It must be able to handle floating constraints.

2.1 Lexical choice within a generation system architecture

Generation systems perform two types of tasks: one conceptual, determining the content of text to generated, and one linguistic, determining the form of that text (McDonald, 1983) (McKeown, 1985). Typically, a generator has two modules, each corresponding to

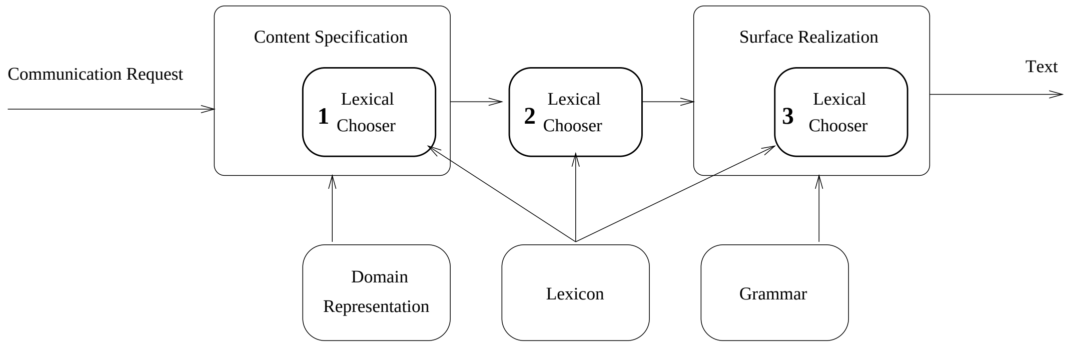


Figure 1
Possible placements of lexical choice within a generator's architecture

one these two tasks, a content planner and a linguistic realizer. While many systems allow for interaction across these components, there is general consensus that these two components can be separated (Reiter, 1994). Furthermore, within the linguistic component, there appears to be further consensus that the task of syntactic realization can be isolated. As evidence, note that a number of dedicated syntactic realization components have been developed such as SURGE (Elhadad and Robin, 1996), NIGEL (Matthiessen, 1991), MUMBLE (Meteer et al., 1987), and TAGs (Yang, McCoy, and Vijay-Shanker, 1991) (Harbusch, 1994). Such components expect as input a specification of the *thematic* structure of the sentence to generate, with the syntactic category and open-class words[3] of each thematic role. Thematic structure involves roles such as `agent`, `patient`, `instrument` etc. It is opposed to surface syntactic structure which involves roles such as `subject`, `object`, `adjunct` etc. Due to general syntactic alternations (Levin, 1993) such as passive, dative, it-extraposition or clefting, the mapping from thematic roles onto surface syntactic roles is one-to-many. The role of the syntactic grammar is to (1) map the thematic structure onto a surface syntactic one, (2) enforce syntactic rules such as agreement, (3) choose the closed-class words, (4) inflect the open-class ones, and (5) linearize the surface syntactic tree into a natural language string. These tasks indicate the kind of information the syntactic grammar needs as input. For example, unless the system is to choose randomly, it needs enough information to choose between different syntactic options available in the grammar. Furthermore, input must either specify all words, or provide enough features so that the syntactic grammar can lexicalize any words that are syntactically determined.

Lexical choice could be carried out at any number of places within this standard architecture. Figure 1 shows the typical language generation architecture used in many systems, indicating the different places for lexical choice to occur. One option would be to position lexical choice as part of syntactic realization, as just a very specific type of syntactic decision (*i.e.,* option 3 in Fig. 1). Researchers who work on reversible grammar formalisms, using the same grammar to both parse and generate language (Van Noord, 1990) (Shieber and Shabes, 1991) (Strzalkowski, 1994), take this approach. The systemic grammar paradigm also takes this approach, where lexical choice is the most "delicate" of decisions, occurring as a by-product of many high-level syntactic choices. However,

3 Words are traditionally divided into (a) open-class words such as nouns, verbs, adjectives and adverbs and (b) closed-class words (also called function words) such as articles, pronouns and conjunctions. Open-classes are large and constantly expanding 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 computational implementations of the systemic paradigm such as NIGEL (Mann and Matthiessen, 1983), only the syntactic constraints on lexical choice are handled during syntactic realization. The semantic constraints on lexical choice are in effect taken into account in the input knowledge representation (*i.e.*, option 1 in Fig. 1).

There are two problems with option 3 (during syntactic realization). First, the range of constraints on lexical choice covered in this line of work is quite restricted and we have some question about whether it could be extended to include the pragmatic constraints considered here. Furthermore, since words are selected only once the full syntactic tree is constructed, it would be quite difficult, if not impossible, to account for floating constraints. Such constraints cannot be considered solely from local positions within a constructed tree, but require some global knowledge of interaction between semantic units.

If lexical choice is not part of the syntactic realization component, then all decisions regarding open-class word selection must be made before the grammar is invoked[4]. They then must occur either as part of content planning or after all content has been determined and expressed in a language independent manner. While some researchers have directly associated words with each concept in the domain knowledge base (*e.g.*, (Reiter, 1991) (Swartout, 1983)), this approach does not allow for consideration of syntactic and lexical constraints unless a phrasal lexicon is used (*e.g.*, (Kukich, 1983b) (Danlos, 1986) (Jacobs, 1985) (Hovy, 1988)). Using a phrasal lexicon, however, means hand encoding the many mappings of multiple constraints onto multi-word phrasings. It thus does not allow for compositional lexical representation (Pustejovski and Boguraev, 1993), and entails a combinatorial explosion in the number of entries to cover the variations of phrases that are possible in different contexts. This approach thus does not allow for scaling up paraphrasing power (see (Robin and McKeown, 1996) for a quantitative evaluation of the scalability gains resulting from the compositional word-based approach).

By waiting until content planning is complete, lexical and syntactic constraints can be represented explicitly and independently of one another, instead of being embedded into full phrases, allowing for a more economical and flexible word-based lexicon which incorporates phrasal constraints.

The only remaining option is to position the lexical choice module between the content planner and the syntactic realization module. Note that some high level decisions about sentence structure must be made early on with this architecture (*i.e.,* before syntactic realization), since, for example, selecting the verb imposes syntactic constraints on how its arguments can be realized. This is desirable since it allows a system to take into account only the type of syntactic constraints on lexical choice that are relevant. In fact, in the eight domains for which we have implemented generators, we have never found a case where other syntactic decisions made during realization force the lexical chooser to undo an earlier decision. This experience strongly supports modularization between lexical choice and syntactic realization.[5]

As we will show, this architecture allows us to convey several aspects of the semantic content using the same word and at the same time, allows us to realize the same semantic concept at a variety of syntactic ranks depending on situation. In particular, by selecting words before a syntactic tree has been constructed, the lexical syntactic features associ-

4 In fact most portable syntactic components mentioned earlier, such as SURGE, MUMBLE, and TAGs, expect as input a fully lexicalized specification, thus supporting this approach.
5 The only argument for position 3 is that it allows for an integrated account of the influence of surface structure onto lexical choice within a single component. However, our experience (corroborated by Reiter's (Reiter, 1994)) shows that the influences on lexical choice from syntax can be accounted for before syntactic realization.

ated with alternate lexical choices can constrain the high level structure of the final tree, which is a key feature to handling floating constraints.

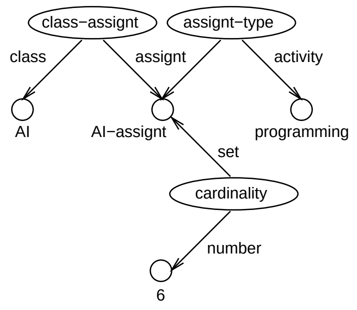
2.2 Input and output

Given this organization, input to the lexical choice module will be structures from the application domain representation which were selected during content planning, possibly augmented with discourse relations. Following our criteria for generation, the structure of the conceptual input will not be committed to any linguistic structure, in order to avoid encoding assumptions about realization into the domain application, and in order to free the content planner from reasoning about linguistic features when determining what information should be included. Thus, it should be possible for a conceptual structure to be realized by a clause, a nominalization, a noun-noun modifier, a predicative adjective or a prepositional phrase. Similarly, the mapping need not be one to one. Several conceptual elements may be realized by the same linguistic constituent, and conversely, several linguistic constituents may be needed to realize a single conceptual element.

For example, consider the conceptual input to ADVISOR-II's Lexical Chooser whose graph is outlined at the top tier of Fig. 2. The domain from which concepts are selected is an expert system rule base, which uses gradual rules of inference (*e.g.,* the more assignments in a class, the harder the class). The input is a set of three relations, each of which is represented similarly by a set of attribute-value pairs in the feature structure form shown in the central tier of Fig. 2, except for cardinality which reduces to an integer. This content representation does not indicate which relations should appear as the head element in the linguistic structure and which should appear as dependents. Nor does it indicate which syntactic relations should be used.

As a result, many different paraphrases of this content can be generated as illustrated by the five given at the bottom tier of Fig. 2. Note that while in (1) the relation `assignt-type` surfaces as the main element in the syntactic structure, in (2-5) it appears as a dependent element. Note also the syntactic variety of these dependent elements. This example illustrates that in contrast to much previous work in generation (but see (Meteer, 1990)) we do not assume that relations will be realized as verbs and objects as their arguments. Instead, the Lexical Chooser must reason about how different domain entities can be realized. The ability to realize relations by compact constituents such as predicative adjectives or noun-noun modifiers allows for the fluency of the sentences of Fig. 2. Realizing all relations in the Fig. 2 input as clauses would result in rather cumbersome sentences such as: "Programming is the kind of assignments of the class whose topic is AI and the number of these assignments is 6."

Note that in order to choose between these sentences, the Lexical Chooser needs information other than just content encoded in the domain representation. In general, the Lexical Chooser needs information about discourse and about speaker intent. For this particular example, it needs information about the speaker's focus and her perspective, at this point in the discourse. Such information must be part of the input to the Lexical Chooser and can typically be provided by a content planner (McKeown, 1985) (Polguère, 1990) (McCoy and Cheng, 1991) (Carcagno and Iordanskaja, 1993) , which must keep track of how focus shifts as it plans the discourse, or text. Similarly, any goals of the speaker must be provided as input to the Lexical Chooser. In the student advising domain, argumentative intent, or the desire of the speaker to cause the hearer to evaluate the information provided in a particular light plays an important role. For example, whether the 6 programming assignments should be viewed as a plus of AI or a minus will depend



$$\left[\begin{array}{ll} \mathit{semr} & \left[\begin{array}{lll} \mathit{assignments} & [1] & \left[\begin{array}{ll} \mathit{cat} & \mathit{set} \\ \mathit{cardinality} & 6 \\ \mathit{generic\_elt} & \left[\begin{array}{ll} \mathit{cat} & \mathit{assignment} \end{array}\right] \end{array}\right] \\ \mathit{class} & [2] & \left[\begin{array}{ll} \mathit{cat} & \mathit{class} \\ \mathit{name} & \mathit{ai} \end{array}\right] \\ \mathit{activity} & [3] & \left[\begin{array}{ll} \mathit{cat} & \mathit{assignment\_activity} \\ \mathit{name} & \mathit{programming} \end{array}\right] \\ \mathit{relation1} & \multicolumn{2}{l}{\left[\begin{array}{ll} \mathit{name} & \mathit{class\_assignt} \\ \mathit{args} & \left[\begin{array}{ll} \mathit{class} & [2] \\ \mathit{assignt} & [1] \end{array}\right] \end{array}\right]} \\ \mathit{relation2} & \multicolumn{2}{l}{\left[\begin{array}{ll} \mathit{name} & \mathit{assignt\_type} \\ \mathit{args} & \left[\begin{array}{ll} \mathit{assignt} & [1] \\ \mathit{activity} & [3] \end{array}\right] \end{array}\right]} \end{array}\right] \end{array}\right]$$

1. "The 6 AI assignments require programming" (relation `assignt_type` as main clause)
2. "AI has 6 assignments which involve programming" (relation `assignt_type` as relative clause)
3. "AI has 6 assignments of programming nature" (relation `assignt_type` as PP)
4. "AI has 6 programming assignments" (relation `assignt_type` as predicative adjective)
5. "AI has 6 implementation assignments" (relation `assignt_type` as noun-noun modifier)

Figure 2
An example of input conceptual network with paraphrases that can be generated from it

both on hearer[6] goals and on what action the speaker[7] thinks the hearer should pursue (*i.e.,* take AI or not). Such goals, or argumentative intent, are used by the content planner in reasoning about what information to include. Again, since such goals are available to the content planner, they can easily be provided as input to the Lexical Chooser.

2.3 Two tasks for lexical choice

Given the input and output specified above, this leaves two tasks for the Lexical Chooser:

- *Syntagmatic decisions*: choosing among the many possible mappings from the flat conceptual network it receives as input onto the thematic tree structure it must produce as output, (*e.g.,* the choice of expressing the `assignt_type` relation in the network of Fig. 2 as the main verb and the `cardinal` relation as a noun modifier in paraphrase (1)).
- *Paradigmatic decisions*: choosing among alternative lexicalizations inside a particular thematic structure, (*e.g.,* the choice of the verb *"to require"* to express the `assignt_type` relation in the paraphrase (1) instead of "to involve" in (2)).

While syntagmatic decisions may seem to be more syntactic in nature, they are directly intertwined with various lexical choices. Selecting the verb, or a higher level relation such as a connective between two clauses, automatically determines overall thematic structure, while selecting which concept in the input will serve as head of the sentence directly influences choice of words. Minimally, syntagmatic decisions include determining the main process[8], which constrains the set of possible verbs. For example, in paraphrase (4) of Fig. 2, this means choosing:

- To map the relation `class_assignt` as the main process of the sentence.
- A `possessive` thematic structure for that main process.
- To map the arguments `class` and `assignt` of `class_assignt` onto respectively the `possessor` and `possessed` roles of the `possessive` process.

Further syntagmatic choices determine which concepts will function as modifiers of any of these roles, ultimately surfacing as relative clauses, prepositional phrases, or adjectival describers. This mapping of conceptual structure to linguistic structure is carried out first in the Lexical Chooser. We call this initial stage involving syntagmatic decisions *phrase planning.*

Then, the Lexical Chooser selects the actual words that are used to realize each role. We call this subsequent stage involving paradigmatic decisions *lexicalization proper.*

Floating constraints are handled in *both* these stages: for example merging two content units in a single linguistic unit is a phrase planning decision, whereas picking the appropriate collocate of an already chosen word is a paradigmatic decision.

6 In our case the system user.
7 In our case the system.
8 Here we use the word "process" in the systemic sense, cf. Sect. 2.4.2.

2.4 An implementation based on the FUF/SURGE package

The implementation of ADVISOR-II builds on a software environment dedicated to the development of language generation systems: the FUF/SURGE package (Elhadad, 1993a) (Elhadad, 1993c). FUF (Functional Unification Formalism) is a programming language based on functional unification (Kay, 1979)[9]. Both the input and the output of a FUF program are feature structures called Functional Descriptions (FDs). The program itself, called a Functional Unification Grammar (FUG), is also a feature structure, but one which contains disjunctions and control annotations. The output FD results from the unification of this FUG with the input FD. The disjunctions in the FUG make unification non-deterministic.

Functional unification has traditionally been used to represent syntactic grammars for sentence generation (*e.g.*, (Appelt, 1983) (McKeown, 1985), (Paris, 1987)) and FUF comes as a package with SURGE, a grammar of English implemented in FUF. SURGE is 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, an 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. Each such knowledge source is represented as a FUG[10]. In the case of ADVISOR-II, the focus of non-syntactic processing is lexical choice. ADVISOR-II thus essentially consists of a pipeline of two FUGs: a lexical FUG encoding the domain-specific lexical chooser and the domain-independent syntactic FUG SURGE. Lexical choice is performed by unifying the conceptual input with the lexical FUG or lexicon. The resulting lexicalized thematic tree is then unified with SURGE which produces the final sentence.

We now briefly overview the FUF language and then the SURGE syntactic grammar before explaining in detail how unification is used to perform lexical choice.

2.4.1 FUF: a functional unification formalism. .

Functional unification is based on two principles: information is encoded in simple and regular structures called functional descriptions (FDs)[11] and FDs can be manipulated through the operation of unification. A Functional Unifier takes as input two FDs and produces a new FD if unification succeeds and failure otherwise. Note that contrary to *structural* unification (SU, as used in Prolog for example), *functional* unification (FU) is not based on order and length (cf. (Shieber, 1992) and (Carpenter, 1992) for recent and comprehensive descriptions of feature structures).

An important property of FDs is their ability to describe structure sharing (or reentrancy): an FD can describe an identity equation between two attributes. For example, subject-verb agreement in a sentence is described by the FD:

$$\begin{bmatrix} subject & \begin{bmatrix} number & [1] \end{bmatrix} \\ verb & \begin{bmatrix} number & [1] \end{bmatrix} \end{bmatrix}$$

In this FD, the notation [1] indicates that the value of the attribute *number* under *subject* must be identical to that of the attribute *number* under *verb*, whatever it may be. In FUF, tags like [1] are encoded with the *path* notation such as `{verb number}`. A

9 FUF is currently implemented in Common Lisp.
10 To avoid overloading the word "grammar", we will use "FUG" to refer to the common representation *formalism* and "syntactic grammar" to refer to syntactic *data* encoded in SURGE using this formalism.
11 FDs are often called feature structures or attribute value matrices in the literature.

path is best understood as a pointer within the FD. Because paths can be used with no constraints, the graph encoding an FD can include loops and does not need to be a tree.

The **`alt`** keyword expresses disjunction in FUF. The value of the **`alt`** keyword is a list of FDs, each one called a *branch*. When unifying an input FD with such a disjunction, the unifier non-deterministically selects one branch that is compatible with the input. Disjunctions encode the available choice points of a system and introduce backtracking in the unification process. In this paper, alts are represented using the following standard notation for disjunctions in feature structures:

$$\left\{\begin{array}{l} \text{ALT name-of-alt} \\ \\ branch1 \\ branch2 \\ \ldots \end{array}\right\}$$

A disjunction can be embedded in another one if necessary. In addition, disjunctions can be named using the **`def-alt notation`**, and referred to in other places using the notation **`(:! name)`**. This notation allows for a modular notation of large grammars written in FUF. Other FUF constructs are introduced as needed in the rest of the paper.

2.4.2 SURGE: a wide-coverage syntactic front-end for generation. SURGE is a wide-coverage syntactic grammar of English implemented in FUF and usable as a syntactic front-end portable across domains. It has been progressively developed over the last seven years and extensively tested for the generation of texts as varied as multimedia explanations (McKeown et al., 1990), stock market reports (Smadja and McKeown, 1991), student advisory sessions (Elhadad, 1993c), telephone planning engineer activity reports (Kukich et al., 1994), (McKeown, Kukich, and Shaw, 1994), taxation correspondence, visual scene descriptions (Abella, 1994), didactic biology term definitions (Lester, 1994), basketball game summaries (Robin, 1994a), workflow diagram descriptions (Passoneau et al., 1996), news article summaries (McKeown and Radev, 1995), intensive care patient summaries (Dalal et al., 1996), and web-page access demographics.

SURGE represents our own synthesis, within a single working system and computational framework, of the descriptive work of several (non-computational) linguists. Our main source of inspiration were: (Halliday, 1985) and (Winograd, 1983) for the overall organization of the grammar and the core of the clause and nominal sub-grammars, (Fawcett, 1987) and (Lyons, 1977) for the semantic aspects of the clause, (Pollard and Sag, 1994) for the treatment of long-distance dependencies and (Quirk et al., 1985) for the many linguistic phenomena not mentioned in other works, yet encountered in many generation application domains. Since many of these sources belong to the systemic linguistic school, SURGE is mostly a functional unification implementation of systemic grammar rules. In particular, the type of FD it accepts as input specifies a "process" in the systemic sense, *i.e.,* any type of situation involving a given set of participants (or thematic roles). This situation can be an event, a relation, or state in addition to a "process" in its most common, aspectually restricted sense. In this broader systemic sense, a "process" is thus a very general concept, simply denoting a semantic class of verbs sharing the same thematic roles. However, SURGE also includes aspects of lexical grammars, such as subcategorization.

Furthermore, while SURGE is essentially systemic in terms of the type of thematic structure it expects as input, it differs from a purely systemic grammar implementation such as NIGEL (Mann and Matthiessen, 1983) in terms of control. Because it is based on functional unification, SURGE is driven by *both* the structure of the grammar *and* that of the input, working in tandem. In contrast, NIGEL is driven *solely* by the structure of the grammar, as encoded in its system networks.

2.4.3 Lexical choice by functional unification. We apply FUF to lexical choice by representing the lexicon as a FUG whose branches specify both constraints on lexical choice (as tests) and the lexical features selected as a result of the different tests. Lexical choice is performed automatically by unifying the lexicon, or lexical FUG, with the conceptual input. During unification, the tests probe both the input conceptual network and the linguistic tree under construction.

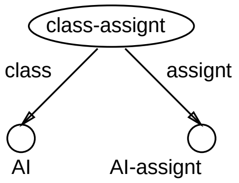
FUF is particularly suited for the representation of lexical constraints for a variety of reasons, some of which have been discussed previously (*e.g.,* (McKeown and Elhadad, 1990), (McKeown et al., 1990)). First, FUF allows the representation of constraints on lexical choice in a declarative and compositional manner. This means that each constraint can be represented separately and the ordering on how the constraints apply is determined dynamically through unification. Because the constraints are represented separately, lexical features are added as each constraint applies, thus compositionally constructing the set of features that define the final choice. Finally, because unification is the governing process, constraints are bidirectional.

Given the wide variety of constraints on lexical choice (Robin, 1990) and the unpredictable manner in which they interact, these features of FUF are particularly desirable. Since in different contexts, different constraints play more or less of a role, unification can determine dynamically which constraints are triggered and in what order. This is a benefit in several different scenarios. For example, sometimes a constraint is stated in the input while at other times it may be derived from the choice of another word in the sentence. If the constraint is available, it can influence the choice of that word; if not, then if the word is selected based on other constraints, it will trigger the constraint which may in turn trigger selection of other words. With lexical constraints that hold between two or more words, it is not critical which word is chosen first. Examples of such patterns of interaction are given in the following sections.

2.5 A simple example

To see how lexicalization works for our simple example sentence *"AI has 6 assignments"* we will only consider semantic constraints. The input specification received by the Lexical Chooser is shown in Fig. 3. The conceptual representation in the input is encoded as an FD under the *semr* (SEMantic Representation) attribute. In this example, it is a simple encoding in FUF notation of a binary relation *class_assignt* holding between two entities: the individual *ai* and the set *assignt_set*1. The link between the arguments of the relation and its fillers is indicated by path values (of [1] and [2] respectively). In the matrix notation used here, we use a number in brackets ([n]) to both label a value and subsequently represent the path to that value. As the lexical chooser proceeds, it adds features to this feature structure, representing the syntactic elements of the clause that is to be produced. The output is shown in Fig. 4. It consists of the input *semr* attribute enriched by a syntactic structure of category clause (cf. note 1) with lexical items specified for each linguistic constituent (cf. note 3 and the features next to note 4 and 5). This output FD is then fed to the SURGE syntactic realization component, to eventually produce the expected sentence.

In the architecture we are describing, the Lexical Chooser must meet the requirements of the underlying application, which feeds it its input, on the one hand, and on the other hand it must produce an output acceptable by the syntactic realization component. The particular semantic features (name of the relations and of the arguments) are specific to the ADVISOR-II domain. The particular thematic roles found in the output



$$\begin{bmatrix} semr & \begin{bmatrix} assignments & [1]\begin{bmatrix} cat & set \\ name & assignt\_set1 \\ cardinality & 6 \\ generic\_elt & \begin{bmatrix} cat & assignment \end{bmatrix} \end{bmatrix} \\ class & [2]\begin{bmatrix} cat & class \\ name & ai \end{bmatrix} \\ relation1 & \begin{bmatrix} name & class\_assignt \\ args & \begin{bmatrix} class & [2] \\ assignt & [1] \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

Figure 3
Conceptual input to the Lexical Chooser in FD format

are characteristic of SURGE. The mapping process is generic.

The Lexical Chooser first traverses the input conceptual structure (which appears under *semr*) to decide what syntactic category will be chosen to realize it. In this case, the lexical chooser decides to map a semantic relation to a simple clause. This decision is done during phrase planning, a process we detail in Sect. 3. As the clause is being constructed, the feature (*cat clause*) is added (cf. Note 1 in Fig 4) and the syntactic structure of a clause is constructed. A clause skeleton is added to the top-level of the FD (cf. notes 2, 4 and 5).

Since the main verb has not yet been selected, the Lexical Chooser cannot proceed further and determine which participants (or verb arguments) will be inserted in the clause, and how they will map to the arguments of the input semantic relation. But the phrase planning component has already determined to use the main verb to realize the input relation. To represent this decision, the Lexical Chooser copies information from the top-level semantic representation in the *semr* feature under *process* (cf. note 2) thus indicating that the main verb maps to the semantic relation *class_assignt*. This is a general feature of the technique: the Lexical Chooser incrementally builds a syntactic structure, and each time a new linguistic constituent is introduced, a subconstituent from the *semr* is copied under the *semr* of the syntactic subconstituent, representing the mapping between semantic and syntactic constituents. This process is the generation counterpart of a compositional semantic interpretation.

The next task of the Lexical Chooser is to select a word or phrase to realize the relation *class_assignt*. The verb is selected by recursively unifying the *process* description (including its newly assigned *semr* feature, cf. note 2) with a disjunction of the verbs stored in the lexicon. The relevant fragment of the lexicon is shown in Fig. 5.

The selected entry[12] contains two types of features: syntagmatic (constraints on daughter nodes in the syntactic tree) and paradigmatic (choice of alternative lexical entries for the same node in the tree). First the verb "to have" is selected (cf. note 3 in Fig. 4) as the default option for possessive verbs - as language uses a possessive metaphor to refer to the link existing between a class and its assignments (a class "owns" the

12 A generation lexicon is indexed by *concepts* instead of words. Each of its entry groups the alternative words to express a given concept.

```
┌                % Tests                                                          ┐
│               ┌ assignments   [1] ... %cf.Fig.3                    ┐             │
│               │ class         [2] ... %cf.Fig.3                    │             │
│               │                                                    │             │
│ semr          │ % Relation1 is of type class_assignt.              │             │
│               │ % It is mapped to the process of the main clause.  │             │
│               │              ┌ name   [3] class_assignt ┐          │             │
│               │ relation1    │ arg1   [1]               │          │             │
│               └              └ arg2   [2]               ┘          ┘             │
│                % Enrichment: semr is mapped to a clause                          │
│                % of process type possessive                                      │
│ cat            clause                                   [Note 1]                 │
│               ┌ semr  ┌ cat    class_relation ┐         [Note 2]  ┐              │
│               │       └ name   [3]            ┘                   │              │
│ process       │ cat   verb_group                                  │              │
│               │ type  possessive                        [Note 3]  │              │
│               └ lex   "have"                                      ┘              │
│                % Indicates the syntactic constituents to be recursively lexicalized
│ lex_cset       ⟨ [4]  [5] ⟩                                                      │
│                                                                                  │
│                % Clause complements                                              │
│               ┌                  ┌ % Pointer to the semr subconstituent ┐ ┐       │
│               │                  │ semr   [1]         [Note 4]          │ │       │
│               │ possessor   [4]  │                                      │ │       │
│               │                  │ % Linguistic features from the lexicon │       │
│               │                  │ cat    proper                        │ │       │
│               │                  └ lex    "AI"                          ┘ │       │
│               │                  ┌ semr          [2]  [Note 5]        ┐   │       │
│ participants  │                  │ cat           np                   │   │       │
│               │                  │ head          ┌ cat   noun         ┐ │ │       │
│               │                  │               └ lex   "assignment" ┘ │ │       │
│               │ possessed   [5]  │ definite      no                   │   │       │
│               │                  │ number        plural               │   │       │
│               │                  │ ref_number    plural               │   │       │
│               │                  │ quantitative  yes                  │   │       │
│               │                  │ exact         yes                  │   │       │
└               └                  └ cardinal      [ value  6 ]         ┘   ┘       ┘
```

Figure 4
Output from the Lexical Chooser

assignments).[13] As discussed below, this is a domain-specific decision that only applies to the particular relation *class_assignt*.

Once the verb class is known, the transitivity of the clause is determined, and the clause skeleton can be extended by specifying the verb's complements. In SURGE, possessive clauses expect two participants, named *possessor* and *possessed*. The second part of the lexical entry therefore determines how the *semr* feature of the two syntactic participants are to be linked to the semantic arguments of the input semantic relation (in our case, this is done by the FUF pointers next to notes 4 and 5 in Fig. 4).

This mapping is domain-specific, and is completely contained in the lexicon entry for the domain relation *class_assignt*. In contrast to previous lexical choice approaches such as (Bateman et al., 1990), we make no claims that the linguistic relation of possession

13 Under different conditions, the lexical chooser could select one of the other verbs represented in the entry, such as *"to require"* or a construction such as *"in* `class`, *there is* `assignment`".

```
% First-level index: relation type is indexed by type of first argument.

{ DEF-ALT RELATIONS
  :index   ⟨process semr cat⟩

   [ process  [ semr  [ cat   class_relation ] ] ]
   [          % This extends into the named alt    ]
   [          % class_relations below              ]
   [ :!       CLASS_RELATIONS                      ]

   [ process  [ semr  [ cat   student_relation ] ] ]
   [ :!       STUDENT_RELATIONS                      ]

              ... Other types of relations ...                }

% Second-level index: relations by name.

{ DEF-ALT CLASS_RELATIONS
   :index   ⟨process semr name⟩

  % For each type of class_relation, determine possible
  % lexicalization and its cat and identify sub-constituents
  [ semr      [ class         [1] ]
  [           [ assignments   [2] ]
  [ process   [ semr  [ name   class_assignt ] ]
  [
  [ { ALT ASSIGNMENTS
  [    :bk_class   ao
  [   % Lexicalizations: C requires A, C has A
  [   % there is A in C, in C they do A
  [
  [   % Branch 1: C requires A
  [   ...
  [
  [   % Branch 2: C has A
  [   [ process        [ type   possessive ]
  [   [                [ lex    "have"     ]
  [   [                % lex_cset declaration to handle recursion
  [   [ lex_cset       ⟨ [3]   [4] ⟩
  [   [ participants   [ possessor  [3] [ semr  [1] ] ]
  [   [                [ possessed  [4] [ semr  [2] ] ]
  [
  [   % Branch 3: In C there is A
  [   [ process         [ type   existential ]
  [   [ lex_cset        ⟨ [5]   [6] ⟩
  [   [ participants    [ located    [5] [ semr   [2] ] ]
  [   [ circumstances   [ location   [6] [ semr   [1] ] ]   } ]

  ... Other types of class_relations ...                           }
```

Figure 5
Fragment of the lexicon for Verb selection

used in this case is more general than the domain relation *class_assignt*. That is, we do not attempt to fit each domain relation under a general ontology based on linguistic generalizations. Such fixed categorization of domain relations in effect prevents a generator from realizing the same domain relation at various linguistic ranks and thus drastically reduces its paraphrasing power[14]. The only information this mapping encodes is that *one* option to lexicalize the domain relation *class_assignt* in English is a possessive clause.

At this point, the top-level unification of the input with the lexical chooser completes. The intermediate FD corresponds to the features next to notes 1, 2, 4 and 5 in Fig. 4. The verb has been selected and the clause structure has been built.

In order to continue the lexicalization of the arguments, the Lexical Chooser must specify which constituents in this FD require further processing. This is accomplished by adding a *lex_cset* (LEXical Constituent SET) declaration to the (lexical) FUG. The value of *lex_cset* is a list pointers to the constituents of the FD as shown in Fig. 5. Constituents bring structure to functional descriptions[15]. To handle constituents, the complete unification procedure implemented in FUF is:

1. Unify top-level input with the lexicon (*ie.,* the single unification step described just above).
2. Identify constituents in result (*i.e.,* read the *lex_cset* attribute).
3. Recursively unify each constituent with the lexicon.

lex_cset declarations appear in the lexical entries for argument-bearing words (mostly verbs). In the example, it specifies the two complements, *possessor* and *possessed*, each of which will eventually be realized as an NP. When this is the case, the head noun of the NP realizes the argument of the conceptual relation. When this argument is shared by other relations in the input conceptual network, those other relations are realized as nominal modifiers. Lexicalizing such complex NPs requires determining:

- Which relations in the complex NP will appear as pre-modifiers and which as post-modifiers.
- Which post-modifiers will be realized as prepositional phrases and which as relative clauses.
- Selecting the features which the grammar needs in order to select a determiner, if any.

Details on how the linguistic features appearing under the NP constituents are selected are given in (Elhadad, 1993b) and (Elhadad, 1996).

In summary, the Lexical Chooser proceeds as follows:

1. A stage of phrase planning first processes the semantic input and determines to which syntactic category it is to be mapped.

14 Note that it is not the very idea of using an ontological upper-model that we criticize here (with all its advantages in terms of knowledge inheritance and reuse) but the use of the most common linguistic realization of each concept as the main criteria for classification.

15 SURGE also uses the special feature *cset* to encode the surface syntactic constituents of the sentence, following (Kay, 1979). Thus, two cross cutting constituent structures, thematic and syntactic, can be represented in the same FD. SURGE ignores the *lex_cset* features and recurses according to the *cset* declarations.

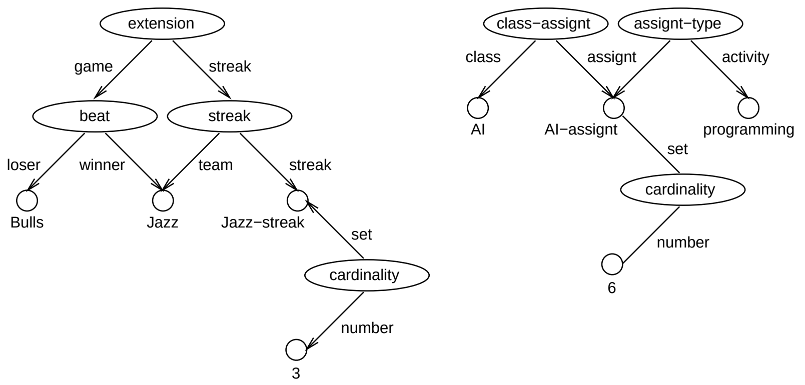


Figure 6
Networks of semantic relations with shared arguments

2. A skeletal FD for the selected category enriches the semantic input.
3. The head word for the linguistic constituent is selected by looking up the semantic feature in (*i.e.,* unifying the *semr* feature with) the lexicon.
4. The lexical entry for the head word is responsible for
   (a) Providing a lexical item with its lexical features.
   (b) Mapping semantic subconstituents to the complements of the head word.
5. The *lex_cset* attribute triggers recursion on the immediate descendants of the linguistic head. The linguistic structure is therefore incrementally expanded as each head is lexicalized in turn. The FUF default control regime develops this structure in breadth-first order.

3. Phrase planning in lexical choice

When the input semantic network contains more than one relation, the Lexical Chooser must decide how to articulate the different predicates into a coherent linguistic structure. We refer to this stage of processing as "phrase planning", because of its close relationship to paragraph planning. In a simplistic generation system, all semantic relations would be mapped to clauses while entity and set descriptions are mapped to noun phrases. This strategy is not felicitous when dealing with multiple relations, as illustrated by the two examples whose inputs and corresponding alternative outputs are shown in figures 6 and 7 respectively on p. 18.

If every relation is to be realized as a clause, then the only option for lexicalizing the relations 1 and 2 in Example 1 is to generate two separate sentences as in (1), or to embed one of the relations as a relative clause modifier of the shared argument as in (2) or (3). Our corpus analysis (Robin and McKeown, 1993), however, has shown that sentences in the basketball report domain tend to be more syntactically complex than sentences (1) to (3). Sentences (4) and (5) illustrate the type of complexity we found: the two semantic relations are merged into a single sentence, but the second relation is realized as prepositional adjunct of different types. Example 2, in the ADVISOR-II domain, shows other options for realizing attached relations in this example: as noun-noun modifiers (*AI assignments*) or pre-modifier (*programming assignments*). To account for this observed syntactic complexity, the Lexical Chooser must be able to accept as

Example 1 (left network of Fig. 6):
(1) Two sentences:
The Jazz defeated the Bulls.
They extended their winning streak to 3 games.
(2) One sentence - `beat` as head - No lexical optimization:
The Jazz who extended their winning streak to 3 games, defeated the Bulls.
(3) One sentence - `streak` as head - No lexical optimization:
The Jazz who defeated the Bulls, extended their winning streak to 3 games.
(4) One sentence - `beat` as head - With lexical optimization:
The Jazz defeated the Bulls for their third straight win.
(5) One sentence - `streak` as head - With lexical optimization:
The Jazz extended their winning streak to 3 games with a victory over the Bulls.

Example 2 (right network of Fig. 6, perspective alternation with fixed focus):
(6) *AI has 6 programming assignments.* (perspective `class_assignt` , focus `AI`)
(7) *The 6 AI assignments require programming.* (perspective `assignt_type` , focus `AI`)

Example 3 (right network of Fig. 6, focus alternation with fixed perspective):
(8) *AI requires 6 programming assignments.* (perspective `class_assignt`, focus `AI`)
(9) *6 programming assignments are required in AI.* (perspective `class_assignt`, focus `assignt_set1`)

Figure 7
Alternative outputs for the input of Fig. 6

input networks of several semantic relations, sharing certain arguments. The semantic networks corresponding to examples 1 and 2 are shown graphically in Fig. 6.

3.1 Selecting the head relation and building its argument structure

The Lexical Chooser must first decide which relation to map to the main clause[16], and which one to embed as a modifier. We refer to this decision as *perspective selection*. The notion of perspective is related to the notion of focus as used, for example, in (McKeown, 1985). However, the perspective is a *relation* in the conceptual network whereas the focus is an *entity*. Once the perspective is chosen, focus can shift between the participants of a relation, by switching the order of the complements, as in sentences (8) and (9) of Fig. 7. This is in contrast to sentences (6) and (7) in the same figure, where perspective switches from `class_assignt` to `assignt_type` (with the focus being the same). We have not investigated further which pragmatic factors affect the selection of perspective. Our research has focused on building into the Lexical Chooser the ability to realize any choice of perspective on the structures produced by the content planner. We thus assume that perspective is given in input to the Lexical Chooser. Figure 2 shows in FD form the input the Lexical Chooser receives for the example which produces the sentences (6) to (9) (the network form for this input was shown at the right in Fig. 6) depending on the values of the additional input features `perspective` and `focus` omitted there.

16 Note that while an object cannot in general serve as a verb, a relation can serve as clause, noun, and a variety of different modifiers. Thus, while we are restricted to selecting a relation as a main clause, we are not restricted in how we do the mapping of other input relations to syntactic constituents.

The ADVISOR-II system expects in its input up to four semantic relations, the highest number of relations that we observed expressed by a single sentence in our model corpus of advising dialogs. The clause planning process has two components: one, domain specific, maps from the domain relations to a clause structure, and one, generic, maps the clause structure to the appropriate types of syntactic modifiers (relative clause, prepositional adjunct, adjectival pre-modifier or noun-noun modifier).

To explain this process, we will step through the clause planning of the example input shown in Fig. 2. Assume that the content planner has decided the focus is on `AI`, and the perspective is on the `class-assignt` relation. First, the head constituent of the linguistic structure is built from the description of the `class-assignt` relation.

This mapping is shown in the top-half of Fig. 8. The left hand side shows the conceptual structure which is the input to the Lexical Chooser. The right hand side shows the linguistic structure that is constructed. At this stage, the conceptual relation which will be realized as the head has been selected (shown by the dotted line pointing to the node `class-assignt`) and the Lexical Chooser has decided to use a process (*i.e.,* ultimately a verb) to realize it. In addition, the `roles` feature is added as a generic argument structure for the clause. The roles point to the appropriate arguments of the `class-assignt` (again, note the dotted lines to the nodes `AI` and `AI-assignt`). Note, however, that during this stage of phrase planning, neither the syntactic category nor the lexical head of the constituent are yet chosen. The input conceptual **graph** is merely transformed into a semantic **tree**. It is only during the subsequent stage of lexicalization (proper), when the specific verb (*"to have"* in this example), is selected.[17] In the implementation, generic roles (role1, role2) are used to point to the arguments of a process as long as the specific verb is not selected. They are mapped to clausal participants (*e.g.*, carrier, attribute) only once the verb is chosen.

3.2 Attaching the Remaining Relations as Modifiers

The second step of the mapping is to map the second relation, `assignt_type` onto modifiers of the arguments of the head clause. The `assignt_type` is mapped onto the modifier slot of the `attribute` role of the head clause, when it is found that the `assignt_type` and `class_assign` share an argument.

This mapping is illustrated in the bottom-half of Fig. 8 for the example sentence:
(1) *"AI has programming assignments"*
The modifier description has the same format as a clause in the linguistic structure. The process of the clause is mapped to the relation `assignt-type` and the process roles to the arguments of `assignt-type`. This does not mean that the modifier will necessarily be realized by a clause as in sentence:
(2) *"AI has assignments which involve programming"*.
It can also be realized by an adjective or a noun. But these modifiers are analyzed as being derived from the relative clause construction using only linguistic derivations, following (Levi, 1978). Thus, sentence (1) above is analyzed as being derived from (2) by deletion of the predicate *"involve"* and migration of its object, *"programming"*, to a pre-modifier of the head *"assignments"*.

Another option to attach a second relation is to add it as a separate clause to avoid deeply embedded structures. For example, the clause combination (sentence 10 below) is preferred over the embedded combination (sentence 11 below), because in the latter the relative clause is twice embedded:

17 The same process generalizes to the treatment of nominal heads.

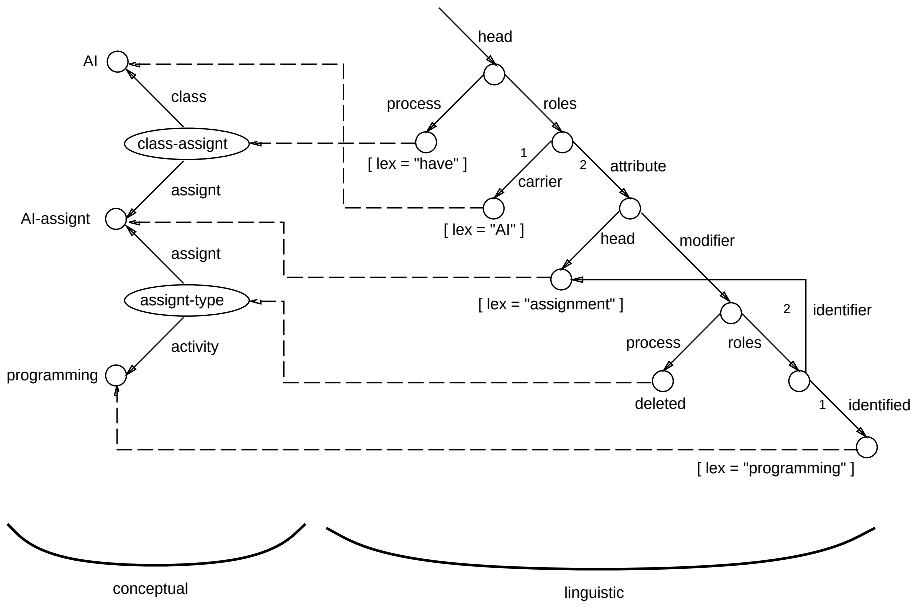


Figure 8
Construction of a clause structure

(10)*Intro to AI has many assignments which consist of writing essays.*
You do not have experience in writing essays.

(11)*Intro to AI has many assignments which consist of writing essays in which*
you do not have experience.

In summary, the first step of the mapping from conceptual network to clause is (1) to select a perspective among the conceptual relations of the network, which determines a head clause, and (2) to attach the remaining relations as either embedded or subordinate modifiers of the head clause. The perspective is selected using focus constraints; the choice between embedding or subordination is based on simple stylistic criteria. The output of this stage is a hierarchical structure where heads correspond to linguistic constituents of a given category (clause or NP), but where the lexical heads are not yet selected.

These two operations constitute *clause planning*, similar to text planning at the paragraph level. A similar process for *NP planning* is described in (Elhadad, 1996). Once the head clause structure has been built, it is passed to the rest of the Lexical Chooser, which determines which syntactic forms can be selected for each modifier, when appropriate lexical resources are found.

These operations of phrase planning are possible in this approach because the conceptual input is not already linguistically structured. Such planning is a major source of paraphrasing power, and since it is controlled by pragmatic factors (as explained in Sect. 4) it also increases the sensitivity of the generator to the situation of enunciation.

4. Cross-ranking and merged realizations

The two structures that the Lexical Chooser has to match - a network of semantic units and a syntactic structure - are in general not isomorphic. This can be explained by two factors: *a combination of semantic elements can be expressed by a single surface element, or a single semantic element by a combination of surface elements* ((Talmy, 1985)

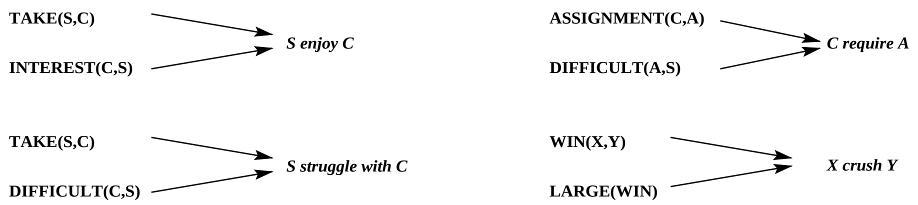


Figure 9
Merging two semantic units onto a single lexical item

1. Judgment determiner: "[AI has **many** assignments.]"
2. Predicative scalar adjective: "[AI, which is **difficult**, has seven assignments.]"
3. Connotative verb: "[AI **requires** seven assignments.]"
4. Argumentative connective: "*AI is interesting* **but** [it has six assignments]."

Figure 10
Cross ranking of argumentative evaluation

p.57). This non-isomorphism between syntactic and semantic structures is a pervasive phenomenon, as illustrated by Talmy's extensive cross-linguistic analysis of constructions expressing motion and causation (cf. (Talmy, 1976) and (Talmy, 1983)).

This discrepancy between the structures of the input and output of the Lexical Chooser imposes two constraints: since several semantic units can be realized by the same lexical item, the Lexical Chooser must be able to *merge* semantic units, and since the same semantic unit can be realized at different syntactic levels, the Lexical Chooser must be able to handle *cross-ranking* realization - that is, to dispatch a semantic unit from a given level in the semantic network onto several different ranks of the syntactic structure.

An example of merging is provided by verbs which convey an evaluative connotation, as illustrated in Fig. 9.[18] Here, the verb *enjoy*, used in *the student enjoyed the AI class*, conveys two semantic units:

- The student took the AI class (a binary relation between the AI class and the student).[19]
- The student found the AI class interesting (an argumentative evaluation).

By choosing a verb with connotations such as *enjoy*, the Lexical Chooser satisfies two input constraints at once.

An example of cross-ranking realization is shown in Fig. 10. All four sentences in this example convey similar information and satisfy the same argumentative intent: they evaluate the AI class as high on the scale of difficulty. The difference is that this evaluation is realized at four distinct syntactic ranks.

18 In which `student`, `class` and `assignments` are abbreviated S, C and A respectively.
19 We do not address the issue of deciding whether posed and presupposed content units will be conveyed in the output.

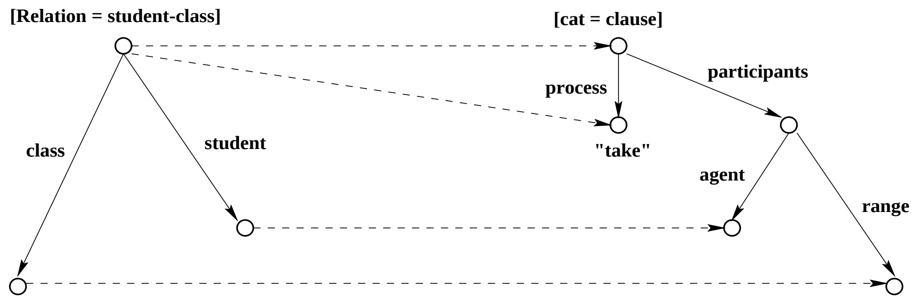


Figure 11
Structural constraint

In (1), the evaluation is realized by selecting the judgment determiner *"many"* and relying on the common sense inference rule "the more assignments in a class, the more difficult it is". Here the Lexical Chooser decided to use the marked evaluative expression *"many"* instead of *"six"* to refer to the number of assignments. Judgment determiners include *"many"*, *"few"*, *"a great number of"* etc. In (2), the use of a scalar adjective directly realizes the evaluative intention of the speaker. Scalar adjectives include *"difficult"*, *"interesting"*, *"important"* etc. In (3), the choice of the connotative main verb *require* can also be related to the speaker's intention to evaluate AI as a difficult class. The verb *"to require"* merges the expression of the relation between the class and the assignments with the connotation that AI is difficult. In contrast, the verb "*to have*" only expresses the first semantic unit and does not have any connotation with respect to difficulty.

Finally, in (4), the argumentative connective *but* projects an evaluation on the clauses it connects in an indirect way. The clause *AI has seven assignments* is not evaluative taken alone; but when it is contrasted with the first evaluation "AI is interesting" by way of a *"but"*, it becomes an argument that must be opposed to "interesting", and the whole sentence supports this second argument. In this case, the speaker relies on two common sense rules that predict that (a) the more a course is interesting, the more a student wants to take it and (b) the more a course is difficult, the less a student wants to take it. Such common sense relations are called *topoi* by Ducrot (Anscombre and Ducrot, 1983). The modeling of argumentative evaluation using topoi for text generation is described in detail in (Elhadad, 1995).

In summary, the same content unit - the evaluation of the class of AI on the scale of difficulty - can be realized by very different linguistic devices: connective, main verb, noun modifier, and determiner sequence. We use the term *floating constraints* to describe input constraints such as evaluations, which can be realized at different syntactic levels. Figures 11 and 12 show how these floating constraints are distinguished from *structural constraints* such as semantic predications. Structural constraints require the presence of syntactic constituents at a given linguistic rank in the output and thus guide the structural mapping process from the conceptual network to the thematic tree. For example, when the input relation `student-class` is mapped to a clause, the predicate *"to take"* determines how the arguments of the relation are mapped onto the thematic roles of the clause: `student` to `agent` and `class` to `range`. In contrast, floating constraints are conveyed by either semantically richer wordings for the obligatory constituents introduced by the structural constraints (*e.g.,* in Fig. 12 changing the verb *"to have"* into *"to require"* or the determiner *"6"* into *"many"* to add the evaluation of AI as a difficult class) or by optional constituents (*e.g.,* in Fig. 12 the connective *"but"* or the qualifier *"difficult"*).

Both merging of semantic units in a single site and cross ranking of a single semantic

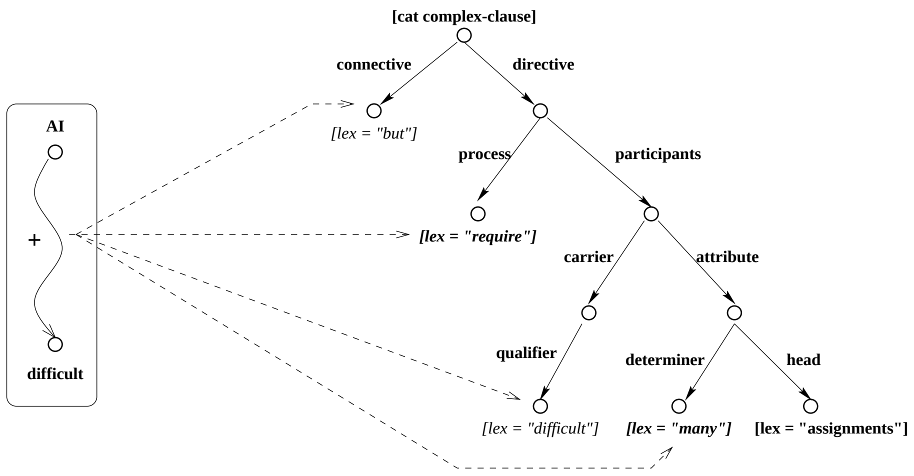


Figure 12
Floating constraint

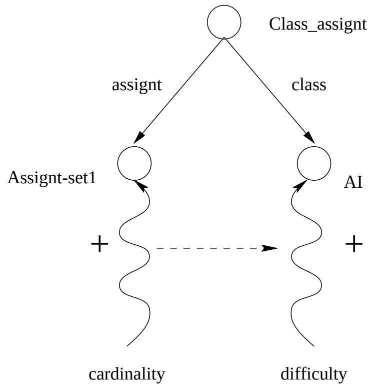
unit to different syntactic sites, are found in other domains as well. For example, in the basketball domain, the semantic unit describing a game result (victory or defeat) can be merged with an evaluation of the ease or the predictability of the result in verbs like *to outlast, to crush, to surprise, etc.* as shown in Fig. 9. Similarly, the phenomenon of cross ranking is not restricted to evaluative connotations. For example, in the following sentences taken from stock market reports (Kukich, 1983b) (Smadja, 1991) the semantic unit expressing the time appears as a floating unit at different syntactic levels:

- Stock prices **got off** to a strong **start**. [time in both (prepositional) verb and object]
- Wall Street Indexes **opened** strongly. [time in verb only]
- Stock indexes surged **at the start of the trading day**. [time in adjunct]

Thus, this phenomenon is a pervasive aspect of lexicalization. The need to perform cross-ranking realization and to deal with floating constraints requires that the input to the generator be *neutral to linguistic form*. This is in sharp contrast with previous generators (Meteer et al., 1987), (Bateman et al., 1990), whose input already determines linguistic structure (*e.g.*, semantic relations are always realized as clauses, and individuals always as noun phrases). The distinction between the structure of the conceptual input and the linguistic structure used to realize it implies that the Lexical Chooser must not only perform paradigmatic choices (select among substitutable items, *e.g.*, between *require* and *necessitate*), but also syntagmatic choices (determine the linguistic structure corresponding to a given input specification, *e.g.*, select between a clause and a nominalization, determine which construction will realize an evaluation). In previous work, these structural decisions were made implicitly at the content determination stage, when building the input to a text generator, and usually were done through hand-coding. In this section, we show how they can be made by the Lexical Chooser in an efficient way.

4.1 Merging of a conceptual unit with an argumentative evaluation

We first illustrate the implementation of floating constraints in a lexical chooser taking as example the merging of a conceptual unit with an argumentative evaluation.



```
[      [            % Evaluative Relation
[      [            [ cat     topos                                              ]
[      [            [         [ scale         cardinality ]                      ]
[      [            [ left    [ orientation   +           ]                      ]
[      [ floating   [         [ value         6           ]                      ]
[      [            [         [ evaluated     [1]         ]                      ]
[      [            [         [ scale         difficulty  ]                      ]
[      [            [ right   [ orientation   +           ]                      ]
[      [            [         [ value         +           ]                      ]
[ semr [            [         [ evaluated     [2]         ]                      ]
[      [
[      [            % Objective relation: AI has 6 assignments
[      [            [                       [ cat           set                      ]  ]
[      [            [ assignments    [1]    [ cardinality   6                        ]  ]
[      [            [                       [ generic_elt   [ cat   assignment ]     ]  ]
[      [ structural [ class          [2]    [ cat    class ]                            ]
[      [            [                       [ name   ai    ]                            ]
[      [            [                 [ name                       class_assignt ]      ]
[      [            [ relation1       [ args [ assignt   [1] ]                   ]      ]
[      [            [                 [      [ class     [2] ]                   ]      ]  ]  ]
```

Figure 13
Lexical chooser input for one semantic unit and one floating constraint,
Example of corresponding output sentence: *"AI has many assignments, so it is difficult."*

Consider the task of mapping an argumentative evaluation onto a simple clause. In this simple example, the input is made up of two semantic units: a conceptual relation, *e.g.*, **`class_assignt(assignt_set1, ai)`** and an argumentative evaluation, *e.g.*, **`eval(ai, difficulty)`**. For this example, we will restrict the available sites in the syntactic clause capable of realizing the evaluation to be: the main verb, modifiers of the NP realizing the class (*i.e.*, pre-modifying adjective or relative clause) and the determiner sequence.[20]

The input description for this configuration of semantic units is shown in Fig. 13 both graphically and as an FD matrix. In the graphical representation, argumentative evaluations are represented as wavy lines, and semantic predications as straight lines. The dotted line indicates that the two evaluations (many assignments and difficulty) are part of an argumentative rule - a topos - which reads: the more a class has assignments the more difficult it is. In the FD matrix, the evaluations and the conceptual relations

20 In the implementation of ADVISOR-II, the Lexical Chooser also considers as potential sites connectives and *indirect argumentative evaluations* - *i.e.*, relying on a topos relation of the form *"the more a class has assignments, the more difficult it is"*, one can realize the evaluation of a class as difficult by evaluating as large the set of assignments it requires. In that case, a modifier of the NP *"assignments"* realizes the evaluation on the entity *class*.

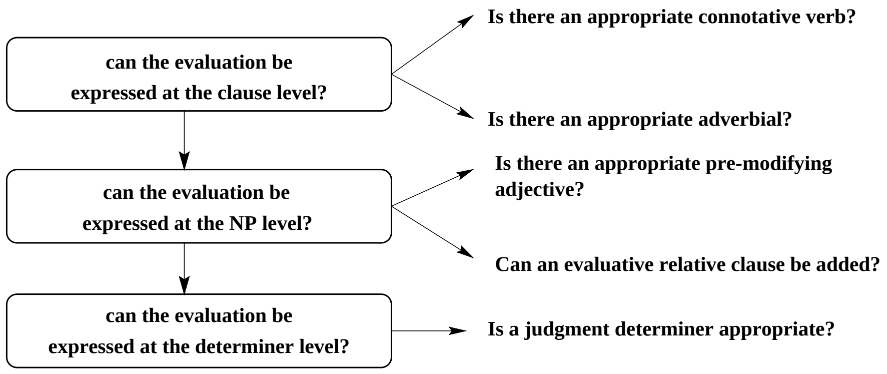

can the evaluation be expressed at the clause level?
Is there an appropriate connotative verb?
Is there an appropriate adverbial?
can the evaluation be expressed at the NP level?
Is there an appropriate pre-modifying adjective?
Can an evaluative relative clause be added?
can the evaluation be expressed at the determiner level?
Is a judgment determiner appropriate?

Figure 14
Order of decisions during top-down traversal for mapping argumentative evaluations

are represented under two separate top-level features of the input `semr` (`floating` and `structural` respectively). This representation does not commit the lexical chooser to map the evaluations to any particular site a priori - highlighting the floating nature of evaluations.

Therefore, one of the tasks of the Lexical Chooser, is to decide to which node in the linguistic tree the argumentative evaluations will be attached. In the overall flow of control followed by FUF (discussed in Sect. 2.5), semantic relations are mapped onto a linguistic tree, which is expanded top-down breadth-first. So the decision to attach an evaluation at a given node in the linguistic structure must also be made top-down. This order of decision is shown in Fig. 14 and is followed as the clause structure is constructed.

At each stage of this traversal, the Lexical Chooser checks whether there is lexical material available in the lexicon to realize the evaluation at the specified syntactic level. For example, at the verb level, the lexical chooser checks whether there exists a verb which expresses the `class_assignt` relation and has for connotation that the `class` argument is difficult. In this case, the verb *"require"* can be selected. If, however, the evaluation was on the scale of interest, no appropriate verb could be found (there is no verb expressing both that a course is interesting and that it involves certain assignments). At the NP level, the Lexical Chooser would then check whether there exists a scalar adjective which can realize the evaluation.

The way these decisions are implemented in FUF is shown in Fig. 15 and 16. The overall process is shown in Fig. 15: the Lexical Chooser first tries to attach the element expressing argumentative evaluation[21] (`AO` attribute in the figure) to an appropriate constituent (in the `map-ao-clause` alt). Then the regular structural constraints are dealt with (in the `roles` alt). The `map-ao-clause` alt implements the following algorithm: if no AO is specified in the input, nothing needs to be done (first branch). Since unification is bi-directional, a feature [a v] in a grammar branch can be either a test for the presence of the feature or the instruction to enrich the input FD with that feature if it is not present. The FUF keyword `given` is used to specify that the feature is intended only as a test. If an AO is specified, then it can be attached either on the process (branch 2) of the clause, or as an adverbial (branch 3). In last recourse, it can be "delegated" to another lower-level constituent (branch 4). Note that at this level of processing, the Lexical Chooser does not yet know whether appropriate lexical material will be found to satisfy the constraint of expressing the AO connotation. For example, when deciding to

21 More often known as speaker intention or goal.

Input semantic net:

$$I0 = \left[\; semr \; \left[\begin{array}{ll} structural & \ldots \%cf.Fig.13 \\ floating & \ldots \%cf.Fig.13 \end{array} \right] \right]$$

has been packaged as follows by the phrase planning branch:

$$I1 = \left[\begin{array}{ll} process & \left[\; semr \quad \ldots \; \right] \\ roles & \left[\begin{array}{ll} 1 & \left[\; semr \quad [1] \;\; \text{\% Under structural} \; \right] \\ 2 & \left[\; semr \quad [2] \;\; \text{\% Under structural} \; \right] \end{array} \right] \\ ao & \left[\; semr \quad [3] \;\; \text{\% Under floating} \; \right] \end{array} \right]$$

```
[ CONJ LEX-CLAUSE
  cat                  clause
  process              [ cat   verb_group ]
  { ALT MAP-AO-CLAUSE
    :bk-class ao

    % At the top-level of the clause, AO can be mapped to:
    % (1) Verb with connotation
    % (2) An adverbial
    % If no lexical resource is found at this level:
    % Try to attach AO at another level

    % Branch 1: No AO specified in input - nothing to do
    [ ao   NONE ]

    % Branch 2: Attach AO down to verb
    [ ao        [4] GIVEN
      process   [ ao   [4] [ conveyed   process ] ] ]

    % Branch 3: Attach AO to adverb
    [ ao        [5] GIVEN
      adverb    [ ao   [5] [ conveyed   adverb ] ] ]

    % Branch 4: Delegate AO to other constituents
    [ ao   GIVEN
      ao   [ conveyed   ANY ] ]
  }

  :! roles ]
```

Figure 15
Lexicon fragments mapping a floating constraint at the clause level

try branch 2, we still do not know whether an appropriate verb will be found. So we just try to attach the AO at a certain level, and wait until the Lexical Chooser arrives at the level of the verb in its traversal of the constituent tree to verify whether our decision was justified. This is the equivalent of a prediction step in a top-down parsing algorithm.

In the end, we must find an appropriate lexical element which will satisfy the requirement to express AO. Such an element is shown in Fig. 16. It is the lexicon entry for the semantic relation `assignments` which holds between a class and its assignments. Three options are listed to realize this relation: `class` *require* `assignment`, `class` *have* `assignment` and *in* `class` *there is* `assignment`. Only the verb *"to require"* conveys an argumentative evaluation (that the class is difficult). When it is selected, the feature [[*ao* [[*conveyed verb* – *level*]]]] is added - which signals that the AO is effectively expressed

In the context of I1 in Fig.15 for the reference to [1] and [2]:

$$\left\{\begin{array}{l}
\text{DEF-ALT ASSIGNMENTS}\\
\quad :\mathit{bk\_class}\quad \mathit{ao}\\
\text{\% Branch 1: C requires A - Argumentative evaluation}\\
\left[\begin{array}{ll}
\mathit{process} & \left[\begin{array}{ll} \mathit{type} & \mathit{attributive}\\ \mathit{lex} & \mathit{``require''}\\ \mathit{ao} & \mathit{GIVEN}\ \text{\% Only if ao is specified in input}\\ \mathit{ao} & \left[\begin{array}{ll} \mathit{evaluated} & [1]\\ \mathit{scale} & \left[\begin{array}{ll}\mathit{name} & \mathit{``difficulty''}\end{array}\right]\\ \mathit{conveyed} & \mathit{verb\_level}\\ \mathit{orientation} & +\end{array}\right]\end{array}\right]\\
\mathit{lex\_cset} & \left\langle [4]\quad [5]\right\rangle\\
\mathit{participants} & \left[\begin{array}{ll} \mathit{carrier} & [4]\left[\begin{array}{ll}\mathit{semr} & [1]\end{array}\right]\\ \mathit{attribute} & [5]\left[\begin{array}{ll}\mathit{semr} & [2]\end{array}\right]\end{array}\right]
\end{array}\right]\\
\text{\% Branch 2: C has A - no AO connotation}\\
\left[\begin{array}{ll}
\mathit{process} & \left[\begin{array}{ll} \mathit{type} & \mathit{possessive}\\ \mathit{lex} & \mathit{``have''}\end{array}\right]\\
 & \text{\% lex\_cset declaration to handle recursion}\\
\mathit{lex\_cset} & \left\langle [6]\quad [7]\right\rangle\\
\mathit{participants} & \left[\begin{array}{ll} \mathit{possessor} & [6]\left[\begin{array}{ll}\mathit{semr} & [1]\end{array}\right]\\ \mathit{possessed} & [7]\left[\begin{array}{ll}\mathit{semr} & [2]\end{array}\right]\end{array}\right]
\end{array}\right]\\
\text{\% Branch 3: In C there is A - no AO connotation}\\
\left[\begin{array}{ll}
\mathit{process} & \left[\begin{array}{ll} \mathit{type} & \mathit{existential}\end{array}\right]\\
\mathit{lex\_cset} & \left\langle [8]\quad [9]\right\rangle\\
\mathit{participants} & \left[\begin{array}{ll} \mathit{located} & [8]\left[\begin{array}{ll}\mathit{semr} & [2]\end{array}\right]\end{array}\right]\\
\mathit{circumstances} & \left[\begin{array}{ll} \mathit{location} & [9]\left[\begin{array}{ll}\mathit{semr} & [1]\end{array}\right]\end{array}\right]
\end{array}\right]
\end{array}\right\}$$

Figure 16
Three alternative lexical entries satisfying an AO constraint

at the verb level.

In summary, the process develops as follows: at the clause level, choose a site for the AO. Here we committed AO to the verb by copying the feature AO from the clause level under the process constituent, with the feature conflation:

$$\left[\begin{array}{ll} \mathit{ao} & [4]\mathit{GIVEN}\\ \mathit{process} & \left[\begin{array}{ll}\mathit{ao} & [4]\end{array}\right]\end{array}\right]$$

Then proceed with regular structural traversal of the constituent tree: process, arguments, then under process, verb. At the verb level, we look up the lexicon for an appropriate entry to realize the process. In this case, the process is already enriched with a copy of the AO annotation. So we select the verb *"to require"*, which satisfies this request. The AO constraint is thus satisfied, and the [*ao conveyed*] feature is instantiated - which means that the AO constraint is satisfied.

Now, imagine that no lexical entries were found to express the AO constraint, neither at the verb level, nor at the adverbial level. In this case, the AO constraint must be mapped down to an NP participant of the clause. The mapping inside the NPs is not described in the `map-ao-clause` alt, because, at this level, the Lexical Chooser has no knowledge of which NPs will be added to the syntactic structure. Instead, the AO constraint is simply delegated to another constituent. When a new NP constituent is constructed by the Lexical Chooser, the alt `map-ao-np` will perform a function similar to `map-ao-clause` but within the NP.

At the clause level, the delegation is encoded using the feature:

$$\left[\ ao \quad \left[\ conveyed \quad ANY\ \right]\ \right]$$

The **ANY** feature is a powerful construct of FUF which imposes that a feature be instantiated *at the end of the unification process*. Because unification is order-independent, **ANY** constraints can only be checked after the entire grammar has been traversed at all the levels. In our example,

$$\left[\ ao \quad \left[\ conveyed \quad ANY\ \right]\ \right]$$

declares that AO must eventually be instantiated to a ground value - that is, that some lexical element has finally taken the responsibility to express the AO.

If we look back at the flowchart in Fig. 14, we see that the algorithm just described deterministically maps the AO to the first slot found in the constituent tree that can account for it. We discuss in the next section how to add variety to this decision and allow a non-deterministic selection of the AO realization site.

4.2 BK-CLASS: smart backtracking for efficient cross-ranking realization

The process just described is a search process where the Lexical Chooser tries to find an appropriate site for the realization of a floating constraint. This search is not tightly constrained, since, for example, at the clause level, the Lexical Chooser maps the AO feature to different sites without knowing in advance whether the sites are capable of handling it. For example, the AO element is first tentatively mapped to the verb, without knowing whether an appropriate connotative verb will eventually be found.

On the other hand, the construction of the linguistic structure depends on this mapping; for example, if the argumentative evaluation is mapped to a qualifier, another type of semantic element will not be mapped to the same site[22]. Thus, it is not possible to first build the whole linguistic structure ignoring the AO constraint and then subsequently examine which sites are available for the AO.

In summary, the problem of finding an appropriate site to attach a floating constraint is inherently a hard search problem, which cannot be handled efficiently by straightforward search techniques. Additional knowledge must be added to control the search and make it efficient. In this section we introduce `bk-class`, a control tool implemented in FUF, which reduces the search space in a significant way and makes the implementation of such lexicalization tasks practical.

`bk-class` is a version of dependency-directed backtracking (de Kleer et al., 1979) specialized to the case of FUF. The `bk-class` construct relies on the fact that in FUF, a failure always occurs because there is a conflict between two values for a certain attribute at a leaf location in the input FD, as already enriched by some features from the FUG at the point of failure within the recursive unification process. In our example, backtracking is triggered by the requirement that the value of the sub-attribute `conveyed` of the attribute `ao` be instantiated, when the actual sub-attribute is not. The path {`ao conveyed`} defines the *address of the failure*, *i.e.,* the only decision points in the backtracking stack that could have caused the failure. Identifying the address of a failure requires additional control knowledge that must be declared in the FUG. More precisely, we allow the FUG writer to declare certain paths in the FUG to be of a certain `bk-class`. We then require the explicit declaration in the FUG of the choice points that correspond to this `bk-class`. In such cases, the writer must realize that the decisions

22 Here, we assume a simplistic stylistic constraint of limitating the number of post-modifiers to one.

within the class are interdependent, but she needs not have knowledge of which one is most likely to succeed.

For example, the statement: `(define-bk-class AO conveyed)` specifies that all paths ending with the attribute `conveyed` are of class AO. In addition, we tag in the FUF program all `alts` that have an influence on the handling of the AO constraint with a declaration `(:bk-class AO)` as shown in Fig. 15 and 16.

To illustrate how `bk-class` helps in dealing with floating constraints, consider again the *require* example discussed in the previous sub-section. In the lexicon fragment of Fig. 15 costly backtracking is avoided by forcing a fixed order for the consideration of the possible sites to express the AO: first verb, then adverb and finally argument NP. Also the alternative option of a separate clause is not considered.

To overcome the limitation of this deterministic site selection for the AO attachment and allow a variety of possible output, we can force a random selection in the `alt map-ao-clause` of Fig. 15. This is achieved by adding a `(:order :random)` annotation to the `alt map-ao-clause`, which becomes:

```
(def-alt map-ao-clause (:bk-class ao) (:order :random) ...)
```

With this annotation, FUF chooses non-deterministically one of the four branches of the disjunction. In this manner, the algorithm shown in Fig. 14 is not performed in a fixed order. As a consequence, the Lexical Chooser can map the AO evaluation to a noun modifier, even though it was possible to merge it into the verb – producing either *AI requires 6 assignments* or *AI, which is difficult, has seven assignments.* In the Lexical Chooser of ADVISOR-II, the AO mapping mechanism also interacts with the phrase planning component and maps the AO to a dedicated evaluative clause - producing in our example the complex clause *AI has seven assignments; therefore it is difficult.* A random selection among these options adds significantly to the variety of possible output.

This randomization can also lead to computationally expensive dead-ends. The AO mapping grammar describes three possible attachment sites for AO, leading to the sentences:

- *AI* **requires** *6 assignments.*
- *AI*, **which is difficult**, *has 6 assignments.*
- *AI has 6 assignments*; **therefore it is difficult.**

If one of these sites is not available, the other one must be chosen. Unfortunately, it can take time to realize that a given site is not available. There are three cases:

- **Case 1: all sites available**: all three sentences can be generated.
- **Case 2: the NP site is not available** because another modifier must be attached to the same NP. This would occur for example in sentences like *AI, which is taught by Smith, has 6 assignments.*
- **Case 3: the verb site is not available** because there is no connotative verb that simultaneously conveys the structural relation and the AO evaluation. This would occur for example in sentences like *AI, which is difficult, deals with some mathematical topics*, since there is no English verb that expresses the relation "deal with" together with a connotation of difficulty in the domain sub-language of ADVISOR-II.
- **Case 4: both the verb site and the NP site are not available** (the combination of case 2 and 3 above): only a complex clause can be generated.

The problem is that at the time the `map-ao-clause` disjunction is traversed, we do not yet know whether the corresponding sites are available. The following scenarios illustrate how the late detection of availability can lead to a prohibitive search time:

- **Case 1: All sites available** The order in which `map-ao-clause` is traversed determines which sentence is generated. No price is paid computationally since all choices lead directly to an acceptable configuration.
- **Case 2: NP site not available - Verb site tried first in map-ao-clause** In this case again, there is no time wasted, the Lexical Chooser "guesses" right at the first time.
- **Case 3: NP site not available - NP site tried first** in this case, the Lexical Chooser goes the wrong way and must backtrack. Backtracking is triggered only when the

  $$\left[\ ao \quad \left[\ conveyed \quad ANY\ \right]\ \right]$$

  constraint is checked, which is *at the end of the unification process.* This means that full lexicalization of the input is performed - ignoring the AO constraint - and at the end, a failure is discovered. Using a simple chronological backtracking control, such a late failure is the worst possible scenario. All possible options starting at the wrong guess in `map-ao-clause` are systematically explored, until finally, the `map-ao-clause` choice point is tried again, leading to an acceptable sentence.
- **Case 4: Two sites (verb and NP) not available - both tried before the complex clause option**. The scenario is the same, except that when backtracking reaches the `map-ao-clause`, it starts once more with a wrong guess, and the same wasteful search is triggered.

The `bk-class` construct solves this problem efficiently by jumping back directly to the `map-ao-clause` when the `ANY` failure is detected at address {`ao conveyed`}. This is possible only if the grammar writer has declared the `map-ao-clause` as a choice point of class `AO`. Figure 17 illustrates the search process, and how `bk-class` affects it. When the unifier fails at a location of class `AO`, it *directly* backtracks to the last choice point of class `AO`, ignoring all intermediate decisions.

In our example, when the `ANY` constraint fails, we directly backtrack to the last AO choice point encountered, which was in the `map-ao-np` disjunction. If no other option is left in the NP, we backtrack up to the alt `map-ao-clause` of Fig. 16. We therefore use the knowledge that *only* the verb, the np or an additional clause can satisfy the AO constraint in a clause to drastically reduce the search space. Thanks to the `bk-class` construct, this knowledge is *locally* expressed at each relevant choice point, retaining the possibility of independently expressing each constraint in the FUF program.

To evaluate the practical effect of `bk-class`, we have measured the number of backtracking points required to lexicalize different clauses illustrating the scenario discussed above. Table 1 summarizes these measurements, performed on the lexical chooser for ADVISOR-II.

The number of backtracking points required to lexicalize each example clause is listed with and without `bk-class`. The first group, with all sites available, indicates the size of the lexicon. The numbers can be interpreted as the number of decisions the

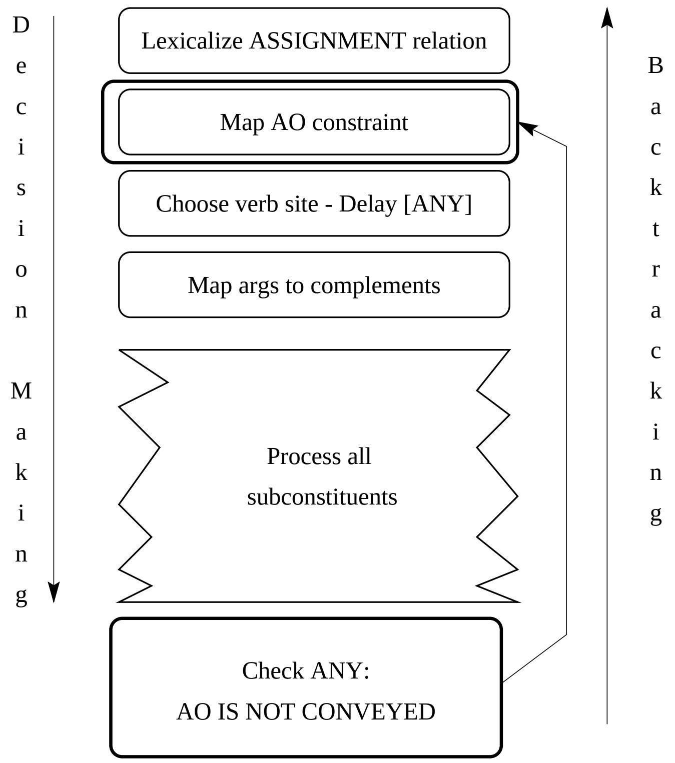

Decision Making
Lexicalize ASSIGNMENT relation
Map AO constraint
Choose verb site - Delay [ANY]
Map args to complements
Process all subconstituents
Check ANY: AO IS NOT CONVEYED
Backtracking

Figure 17
Backtracking and bk-class

Lexical Chooser makes to lexicalize a basic clause for which practically no backtracking is required. It roughly corresponds to the number of unretracted decisions made by the Lexical Chooser and is the optimal number of backtracking points that a search control regime can obtain for the given input. Without `bk-class`, the wide variation in number of backtracking points among the examples indicates the exponential nature of the blind search which floating constraints impose on the standard control regime. In contrast, with `bk-class`, the variation in number of backtracking points remains within a factor of three among all the examples.

The dependency-directed mechanism implemented in FUF with `bk-class` therefore complements a general top-down control regime to make the processing of floating constraints efficient. The performance penalty imposed by a floating constraint depends on the number of sites in the syntactic structure where it can be realized. For example, the AO constraint can be realized at three levels and it may require the unifier to re-traverse the same FUG branches three times until it finds a site to convey the AO constraint. Each floating constraint can be characterized by its range of possible attachment nodes.

The `bk-class` mechanism improves the efficiency of functional unification while preserving its desirable properties - declarativeness and bidirectional constraint satisfaction. It is a declarative statement of dependency between a decision in the FUG and a class of constraints in the input. Using `bk-class`, however, is not always easy for the FUG writer since it requires thinking about the control strategy of the unifier - the same drawback as for PROLOG's `cut` mechanism. We are currently investigating the use of abstract interpretation techniques (Cousot and Cousot, 1977) to automatically determine where `bk-class` annotations are necessary and thus ease the task of the programmer.

Table 1
Measuring the effect of `bk-class`

		Backtrack pts	
Input	Output	w/o `bk`	w/ `bk`
All sites available	*AI requires 6 assignt..*	46	46
	AI, which is hard, has 6 assignt..	59	59
	AI has 6 assignt.., therefore it is hard.	189	189
Verb unavailable			
NP site tried first	*AI, which is hard, deals with math.*	58	58
2 clauses tried first	*AI deals with math, therefore it is hard.*	59	59
Verb site tried first	*AI, which is hard, deals with math.*	**8,379**	**86**
NP unavailable			
NP site tried first	*AI, ..., requires 6 assignt..*	**10,546**	**124**
NP and verb unavailable	*AI, ..., deals with math, therefore it is hard.*	**15,580**	**128**
2 clauses tried last	*AI, ..., has 6 assignt.., therefore it is hard.*	**22,719**	**189**

5. Inter-lexical constraints

Inter-lexical constraints occur when a pair of content units is realizable by alternative sets of collocations (Smadja and McKeown, 1991). This is the case, for example, in the following situation:

- A domain relation R is realizable by two verbs $V1$ and $V2$
- A domain entity E is realizable by two nouns $N1$ and $N2$.
- $< V1, N1 >$ and $< V2, N2 >$ are verb-object collocations.
- $< V1, N2 >$ and $< V2, N1 >$ are *not* verb-object collocations.

We observed the influence of inter-lexical constraints in our corpus of newswire basketball reports used in developing STREAK. The act of performing strictly better than ever before can alternatively be lexicalized by the collocations *"to break a record"* or *"to post a high"*. However, even though the elements of the collocation are dependent on different portions of the input network, they are not interchangeable. One can say neither *? "to break a high"* nor *? "to post a record"*. For other relations, the words are interchangeable; for example, one can say either *"to equal a high"* or *"to equal a record"*.

The input for these collocations is shown in Fig. 18. It includes two relations: **`performance(player, statistic)`** and **`improve(performance, maximum)`** which indicate that a player's performance improves some previously set maximum. The verb of the collocation realizes these two semantic relations, such as *"to break"* or *"to post"* simultaneously indicating (1) a relation between a player and one of its game statistics and (2) that this statistic is strictly higher than some previous maximum. If, in contrast, the statistic was even with the previous maximum, then the verb *"equal"* or *"match"* would be selected instead. Whether the object is *"high"* or *"record"* depends on the type of maximum. In this domain, a maximum can occur for a variety of different reference sets. It can be a maximum for a player, for a team, or for a league. The noun *"high"* is used when the performance is a maximum of past performances by the same player while *"record"* is used when it is maximum over past performances of other players as well.

These semantic constraints on individual word choice are orthogonal to the inter-lexical constraints between verb and object. It is possible, therefore, to encode inter-lexical constraints in various locations in the lexical chooser. One possibility is to encode

```
                          % Relation 1 marked as structural by the content planner
                          % Stockton scored 45 points
semr  structural   relation1  name   performance
                              args   actor  [1]  cat      player
                                                 name     Stockton
                                                 surname  John
                                     perf   [2]  cat      statistic
                                                 value    45
                                                 unit     points

                          % Relation 2 marked as floating by the content planner
                          % This performance improves Stockton's maximum
      floating     relation2  name   improve
                              args   new_perf   [2]
                                     past_perf  [3]


                     % Past performance that the new performance [2] improves:
                     % Maximum of the set of all scoring+ performances scored by Stockton
   [3]   cat       maximum
         ref_set   cat          set
                   generic_elt  name    performance
                                actor   [1]
                                perf    cat   statistic
                                        unit  points
```

Figure 18
Test input for the sentence *"John Stockton posted a high with 45 points"*

the verb-object collocation with the lexicalization options for the predicate. This means that both the lexicalization of the predicate and that of its collocationally constrained argument are chosen all at once. Such simultaneous choices hinder the modularity of the Lexical Chooser and declarative representation of individual constraints. With this organization, the choice of verb must be indexed by the semantic constraints from the arguments even though these constraints do not directly influence predicate choice. Furthermore, if this constraint is encoded in the lexical entry for the predicate, it will have to be repeated for the other domain relations that can take the same concept as an argument such as `equal`. This is undesirable since the verbs for lexicalizing this relation are not collocationally restricted as indicated by the validity of all the following forms: *"to equal a high", "to match a high", "to equal a record", "to match a record".*

A second possibility is to represent the semantic constraints on verb and noun choice separately; where there are several possible verbs for a given semantic constraint, the verb is chosen randomly. The collocational constraints between verb and noun are represented with the noun in this scheme. In this case, modularity in the representation of individual constraints is preserved. Semantic constraints on the respective lexicalizations of the predicate and its argument are independently encoded. However, the orthogonal inter-lexical constraint can trigger backtracking: for example, if the verb *"to break"* is first randomly chosen for the relation `improve` and then the noun *"high"* gets selected because the input indicates the player as the reference set for the maximum concept, the Lexical Chooser must backtrack to select the verb *"post"* instead.

In order to both preserve modularity and avoid backtracking, the solution is to delay the choice of one collocate until the other one is chosen. We have implemented this in FUF through a control mechanism termed `:wait`, which allows explicit representation of the *inter-lexical* constraints along with modular representation of the individual semantic constraints on word choice. `:wait` is used to indicate that a particular choice should be delayed until some other specific choice point in the grammar. Again, the grammar writer must know that the two decisions are interdependent, but does not need to know which one will take priority. Here, the choice of verb is delayed until the head of the object is selected. Figure 19 shows how the `:wait` annotation of FUF implements such a delay. In this case, the choice of verb is suspended until its object noun collocates have been chosen. Therefore, no backtracking occurs even though the respective semantic constraints on the lexicalization of the predicate and its argument are kept separate. The `:wait` annotation is a general control facility that allows optimizing a FUG whenever it is known that two decision classes are dependent on one another. The case illustrated in this paper, where these two decisions classes are verb choice and object head noun choice is only one of the many types of optimizations made possible by the use of `:wait` (see (Elhadad, 1993c) for other types).

6. Other approaches to lexical choice

6.1 FUF based systems

In this section, we present four applications developed at Columbia which use FUF for lexical choice. The first two use a very similar approach to ADVISOR-II, while the last two incorporate the same model within distinct system architectures.

6.1.1 Systems with similar architectures. Two applications developed at Columbia use FUF for lexical choice in much the same way as ADVISOR-II: COMET (COordinated Multimedia Explanation Testbed) (Feiner and McKeown, 1990) (Elhadad et al., 1989) and PLANDOC (McKeown, Robin, and Kukich, 1995). These systems are more limited, however, in that they do not handle floating constraints, and thus lexical choice consists of a one-to-one mapping between conceptual and linguistic structures. This mapping focuses on the paradigmatic decisions and can be efficiently performed using the simple default top-down regime of recursive unification. But in effect, this shifts the burden of performing most syntagmatic decisions to the Content Planner which must then hand to the Lexical Chooser a tree-structured input that already prefigures the (lexicalized) thematic tree that the Lexical Chooser in turn passes on to SURGE.

In COMET, a system which generates multimedia explanations for equipment maintenance and repair, the major research focus is the coordination of multiple media and some word choices are influenced by decisions made by the graphics component. For example, to determine how to refer to an object in an accompanying illustration, the Lexical Chooser takes into account how the object is depicted (McKeown et al., 1992). COMET also considers constraints from the user and from previous discourse in selecting words. This case has some similarities with floating constraints; if a word is unknown to the user, an alternative word may not exist which can simply be substituted in the sentence. Instead, COMET must replan the entire sentence when an alternative word is not available, reinvoking its content planner (McKeown, Robin, and Tanenblatt, 1993).

PLANDOC is an automated documentation system under joint development by Columbia and Bellcore. It produces 1-2 page reports documenting the activities of telephone planners. While PLANDOC does include a wide variety of paraphrasing, some of which involve different word choices, and handles interactions between different choices, lexical choice was not the primary issue in this system. Instead, the focus was on the systematic use of conjunction with ellipsis and its interaction with syntagmatic paraphrases, in order to produce concise, yet fluent summaries. FUF was used as a tool for developing the lexical chooser, but with less novel results on the topic of lexical choice.

6.1.2 Systems with different architectures. Two other applications developed at Columbia also use FUF for lexical choice, but within a different system architecture:

```
{ DEF-ALT LEX_SCORE

  % Branch 1: . . .

  % Branch 2: Verbs merging a PERFORMANCE structural relation
  % and an IMPROVE floating relation
  % Unifies with result of clause planning on input of Fig.18
  [ process    [ semr [ name  performance ] ]
    roles      [ actor  GIVEN
                 perf   [1] GIVEN ]
    floating   GIVEN
    floating   [ process [ semr [ name  improve ] ]
                 roles   [ new_perf  [1]
                           old_perf  [ cat  maximum ] ] ]
    :!         IMPROVE_MAX_VERBS                              ] }


{ DEF-ALT IMPROVE_MAX_VERBS
  % Choice of verb delayed until object is lexicalized
   : order   random
   : wait  ⟨ participants affected head lex ⟩

  % Purely lexical test since done when object noun already chosen
  % Branch 1: break a record
  [ process        [ lex  "break" ]
    participants   [ agent     [1]
                     affected  [2] [ head [ lex  "record" ] ] ]
                   % Recurse on both arguments
    lex_cset       ⟨ [1]  [2] ⟩                                ]

  % Branch 2: post a high
  [ process        [ lex  "post" ]
    participants   [ agent     [3]
                     affected  [4] [ head [ lex  "high" ] ] ]
                   % Recurse on both arguments
    lex_cset       ⟨ [3]  [4] ⟩                                ] }


[ DEF-CONJ LEX-MAX

  semr  [ cat  maximum ]
  cat   np

  { ALT SEMANTIC-CONSTRAINT-ON-NOUN
    % Choose noun for max. with no reference to embedding verb
    % Branch 1: record
    [ semr  [ ref_set [ generic_elt [ actor  [ cat {team league} ] ] ] ]
      head  [ lex  "record" ]                                         ]

    % Branch 2: high
    [ semr  [ ref_set [ generic_elt [ actor  [cat player] ] ] ]
      head  [ lex  "high" ]                                   ] } ]
```

Figure 19
Use of `:wait` to preserve modularity and avoid backtracking

COOK (Smadja and McKeown, 1991) and STREAK (Robin, 1994b). Some of the examples discussed in this paper within the framework defined by the architecture of ADVISOR-II originated from the respective domains for which these two other systems were implemented.

The focus in COOK, a system for generating stock market reports, was on inter-lexical constraints such as those presented in Sect. 5. By representing in FUF collocations that were automatically derived from a corpus of stock market reports, COOK could merge phrasal and single word constraints in the same lexicon as well as handle interactions between collocations. The way lexical choice is performed in COOK differs from the way it is performed in all the other FUF-based systems in that it is not driven by a top-down traversal of the input conceptual structure. Instead, lexical entries (individual words or collocations) are chosen *in a fixed order* in terms of the syntactic function they will ultimately occupy in the output sentence: first the verb arguments, then the main verb, finally the adjuncts. This bottom-up regime is less efficient than the top-down regime for handling the structural constraints described in Sect. 4. However, it allows indexing the lexicon by <lexical item, syntactic function> pairs rather than by concepts, a property that is handy for using syntactically marked collocations automatically produced by a domain-independent tool with no semantic knowledge such as XTRACT (Smadja, 1991).

The focus in STREAK was the definition of a draft and revision architecture for generating the type of very complex sentences used in newswire reports to summarize quantitative data about a basketball game and its historical background. STREAK relies on revision rules drawn from a corpus analysis of such reports (Robin and McKeown, 1993) to incrementally generate such complex sentences. While the draft and revision approach of STREAK sets it apart from all the other FUF-based generators, the Lexical Chooser of STREAK is akin to the Lexical Chooser of ADVISOR-II in that:

- It handles floating constraints.
- Its input is a linguistically unbiased flat semantic network.
- It is based on top-down recursive functional unification.

There is, however, an important difference in the architectures of the two systems. STREAK handles the structural constraints and the floating constraints in two separate passes. Structural constraints are handled during an initial draft building pass and floating constraints during a subsequent revision pass. This is in contrast with ADVISOR-II which handles both types of constraints simultaneously. In STREAK, the Lexical Chooser is thus called first to build the draft and then again at each revision increment. The need for revision in STREAK primarily stems from the necessity in written summaries to concisely pack many facts in very complex sentences, as observed in newswire stories. For example, in the corpus of basketball reports that served as model output for STREAK some sentences conveyed up to 12 conceptual relations and contained up to 46 words. In contrast, in the corpus of student advising sessions that served as model output for ADVISOR-II no sentences conveyed more than four conceptual relations or contained more than 25 words. For the sentences of more ordinary complexity found in dialogs, revision is not needed and simultaneously combining semantic units into a single word can be achieved far more efficiently in a single pass.

6.2 Other systems

In this section we overview approaches to lexical choice which do not rely on a FUF-based lexicon. We have classified them according to where they position the task of lexical choice in the overall generation architecture. For each class of systems, we describe the constraints on lexical choice that were considered.

6.2.1 After content planning and before syntactic realization. This approach is the most common among text generation systems. The surface realization component is separated into two successive modules: lexical choice followed by syntactic realization. The lexical choice module builds the linguistic structure and chooses all open-class words. The syntactic realization component deals with agreements, ordering of constituents, choice of closed-class words, and in most systems can also perform syntactic transformations on the structure provided by the Lexical Chooser (like passivization, dative move, there-insertion or it-extraposition). This is the approach used in the systems TEXT (McKeown, 1985), GENARO-MUMBLE (Conklin, 1983) (Meteer et al., 1987), PENMAN (group, 1989), SPOKESMAN (Meteer, 1990), EPICURE (Dale, 1992), KALIPSOS (Nogier, 1990) and in the generators based on the Meaning-Text Theory (MTT) (Mel'cuk and Pertsov, 1987) such as: FOG (Bourbeau et al., 1990), GOSSIP (Carcagno and Iordanskaja, 1993) and LFS (Iordanskaja et al., 1994).

These systems primarily use semantic and syntactic constraints in selecting the words to use. Typically, each semantic concept has an entry in the lexicon. The lexical entry for the semantic head (usually the verb, but indicated by either the semantic representation or by the underlying application) is accessed first. The word chosen can depend on the semantic features of the arguments to the head (thus, this scheme basically follows Goldman's use of discrimination networks (Goldman, 1975)). It is at this point that the overall syntactic structure is determined. This lexical entry then usually invokes the lexical entries for the arguments to the head and this control is followed until all input arguments have been lexicalized. In these systems, pragmatic constraints are not consistently accounted for and when inter-lexical constraints are accounted for they are encoded as phrasal entries. Perspective on the input conceptual structure is fixed and thus, different entry points into this input structure cannot determine what the syntactic head of the sentence will be (except in SPOKESMAN (Meteer, 1990)); in other words, the conceptual structure determines linguistic structure. Lexical choice was not a major research issue in any of these systems unlike the work presented here, with two exceptions: KALIPSOS and MTT-based generators.

Research for KALIPSOS focused on mapping multiple conceptual elements to the same word using matching of conceptual graphs. It also allowed for some decision making in determining the perspective of the clause, selecting the verb which covers the largest portion of the input network. Unlike our work, it handled primarily semantic and syntactic constraints on choice and thus, for example, does not dispatch conceptual nodes to different syntactic ranks.

Paraphrasing power using a word-based lexicon is also a central issue in MTT, a generation-oriented and highly stratificational linguistic theory in which sentences are represented at multiple layers, five of which are are relevant to the task of lexical choice (Polguère, 1990).

ADVISOR-II and MTT-based systems are similar in that lexical choice in both starts from a flat conceptual network (the Conceptual Communicative Representation (CCR) layer in the case of MTT-based systems), they perform choice of perspective as well as syntagmatic choices, and they take into account inter-lexical constraints.

There are also three main differences:

- *The MTT approach is more stratified.* In GOSSIP and LFS, lexical choice is decomposed into a pipeline of four mappings between the five layers of MTT representations. In ADVISOR-II, the input flat conceptual network

1. `Main Verb with Adjunct`: "Prices **increased by 20%**"
2. `Object Noun with Post-Modifier`: "Prices showed an **increase of 20%**"
3. `Subject Noun with Pre-modifier`: "A **20% increase** has been reported"

Figure 20
Cross-ranking paraphrasing in the Meaning-Text Theory

(corresponding to the CCR in MTT) is directly mapped onto the output lexicalized syntactic tree[23].

- *MTT makes the different types of lexical choices in a fixed order*: first paradigmatic choices that depend only on semantic constraints, then perspective choice, followed by syntagmatic choices and finally paradigmatic choices that depend on inter-lexical constraints. ADVISOR-II starts with perspective choice and then interleaves syntagmatic and paradigmatic choices.
- *MTT does not handle pragmatic constraints which float across the whole spectrum of linguistic ranks* (*i.e.,* all the way from connectives linking several clauses or sentences down to determiners) such as the argumentative evaluations generated by ADVISOR-II. MTT also does not explicitly distinguish between structural and floating constraints, which makes considering the connective or determiner as alternative sites to clausal or nominal sites problematic. This is significant since using a connective or a determiner instead of a semantically rich verb or a noun modifier allows for a more implicit expression of floating constraints (at least in the case of argumentative evaluations). MTT does, however, allow cross-ranking paraphrasing limited to the clause and NP ranks as illustrated by the examples in Fig. 20 (taken from (Boyer and Lapalme, 1985)).

6.2.2 During surface realization, interleaved with syntactic realization. In this approach lexical choice is considered as one linguistic decision like any other one, and, therefore, it is not isolated in a dedicated component. Instead, the syntactic grammar contains very specific rules for lexical insertion.

This approach is generally associated with the use of either:

- *A phrasal lexicon* such as in the generation systems ANA (Kukich, 1983a), PHRED (Jacobs, 1985) and PAULINE (Hovy, 1988).
- *A lexicalist reversible grammar* (Strzalkowski, 1994), such as a synchronous TAG (Shieber and Shabes, 1991), especially in conjunction with the semantic-head driven algorithm to generation (Shieber et al., 1990).

In a phrasal lexicon entry, all the syntactic constraints between the elements of the template are already pre-selected, so there is no need for much syntactic realization:

23 It is interesting to note that the most applied among MTT-systems, FOG, directly maps the CCR onto a DSyntR. It may be an indication that all the intermediary representations defined in the MTT are not necessary in all domains.

constituents are already ordered, their syntactic category is fixed in the phrasal template, closed class words are already selected. Alternations like passivization or dative-move are encoded by defining, for each domain concept, several different templates, one for each combination of these orthogonal options (and thus failing to capture the generality of these syntactic transformations independently of specific words and concepts). Inter-lexical constraints pose no problem since mutually constrained words are not chosen individually but all at once by accessing a phrasal template (it is just a matter of encoding only the entries corresponding to valid collocations). Similarly, floating constraints are less of a problem for the phrasal approach, since most alternative locations for their expression remain for the most part within the scope of a phrasal template. However, the price to pay for this simplicity is high: generators relying on a phrasal lexicon simply do not scale up. Extending their paraphrasing power and semantic coverage requires hand-coding a combinatorially explosive number of phrasal templates. With a more compositional approach, such scaling up can be achieved by adding only few words along with the constraints that bind some of them (see (Robin and McKeown, 1996) for a corpus-based quantitative evaluation of the scalability gains achievable through compositionality).

In most approaches based on a lexicalist grammar, the semantic structure is traversed and a semantic head is selected; after the head is selected, all the syntactic decisions that depend on this head are performed; then the dependent constituents are identified and lexicalized in turn. Lexicalization and syntactic realization are, therefore, interleaved as the linguistic structure is being built. With this approach, only a restricted range of constraints on lexical choice can easily be handled: in most cases only the semantic and syntactic constraints coming from the local constituent influence what word is selected. Inter-lexical and pragmatic constraints are not discussed and we have some doubts about how easy it would be to incorporate them in this approach. Since words are selected only as the full syntactic tree is constructed, it would be difficult to take floating constraints into account, which collapse different portions of the semantic input into different portions of the syntactic tree.

6.2.3 During content planning and before syntactic realization. Another class of generation systems positions the task of lexical choice as occurring somewhere during the process of deciding what to say, before the surface generator is invoked. This positioning allows lexical choice to influence content and to drive syntactic choice. Danlos (Danlos, 1986) chooses this ordering of decisions for her domain. In her system, a *lexicon grammar* first selects lexical items for the predicative elements of the conceptual input. A *discourse grammar* is then used to combine clauses into valid discourse organization patterns which also determine syntactic realization (choice passive/active for example). Danlos's system thus performs content determination and lexical choice *before* discourse organization and syntactic realization (a very original position). Since all the possible combinations of alternative discourse organizations and alternative syntactic forms need to be hand-coded in the discourse grammar, this approach suffers from the same drawback as the phrasal lexicon approach: it is not compositional and thus does not scale up.

Rubinoff's IGEN (Rubinoff, 1992) also addresses the problem of the interaction between content decisions and linguistic decisions and implements an architecture that relies on explicit negotiation between the two components: the content planner requests a set of options from the Lexical Chooser to realize a certain conceptual element, and the Lexical Chooser replies with annotated options. The content planner selects among these annotations as it proceeds and combines the preferred options into an English message. Rubinoff designed a language of annotations that maintains a good separation of knowledge between the conceptual and linguistic components.

Other researchers advocate folding the lexicon into the knowledge representation. In this approach, as soon as a concept is selected for the text, the words associated with it in the knowledge base would automatically be selected as well. This is the approach in KING (Jacobs, 1987) and FN (Reiter, 1991). In this approach, inter-lexical and syntactic constraints on lexical choice are not addressed, thus restricting the coverage of the domain sub-language of the generator to sentences where these constraints do not come into play. McDonald's (McDonald, 1983) use of realization classes allows for the advantages of this approach while nonetheless allowing for syntactic constraints to play a role. He makes use of inheritance within a knowledge base (McDonald, 1981) to associate generic concepts such as `OBJECT` directly with a phrase category such as `NP`, but allows for individuations of the concept to have exceptions. Options are expressed in realization classes. While this does allow for a given input semantic element to be mapped to different syntactic constructions, this approach does not address the problem of merging semantic concepts in a single word, not does it address interlexical constraints.

7. Conclusion

In this paper, we have presented a general model for lexical choice, which allows a generation system to take into account a wide range of constraints on word selection in a compositional, flexible and yet efficient fashion. By positioning the Lexical Chooser after content selection and before syntactic realization, both conceptual and linguistic constraints can influence word choice. Critically, our work has focused on the problem of floating constraints, providing a mechanism which allows choice of a single word to realize several semantic concepts and conversely allows a single semantic concept to be realized by multiple words at distinct linguistic ranks. Our technique for lexical choice is characterized by the following features:

- It is capable of handling a wide variety of constraints on lexical selection (co-occurrence restrictions, connotation of lexical items, syntactic and conceptual properties and pragmatic effects).
- Interaction among these constraints is easily handled through the use of a unification formalism to describe lexical entries.
- Floating constraints can be attached at different levels of the generated syntactic structure, and several conceptual elements can be merged onto a single linguistic item, providing for more compact and lexically richer output.
- Syntagmatic decisions are also made by the Lexical Chooser, so that an input conceptual structure can be mapped to a variety of linguistic structures. For example, the selection of "perspective" in a clause is guided by the available lexical resources (which verbs exist to express a given conceptual relation) and by pragmatic considerations (which relation is highlighted as the topic of the discourse).

Our implementation of the Lexical Chooser as a functional unification grammar allows the separate, declarative representation of different constraints, with unification allowing for interaction between constraints. Furthermore, within this framework we have developed an algorithm for lexical selection, and the consequent building of syntactic structure, such that at each choice point in the structure, the Lexical Chooser considers all semantic concepts that can be realized, choosing a word that conveys as many as possible or discharging a concept to dependent linguistic sites, at the end checking that all concepts have been covered. In this framework, floating constraints can be merged with other semantic constraints at any linguistic rank.

Finally, handling floating constraints requires that the same semantic structure is mapped to different syntactic structures in different contexts. Thus, a Lexical Chooser must also consider syntagmatic choice. We have presented an approach that allows perspective to be chosen depending on the discourse focus. Different relations may be chosen as the head of a syntactic structure, depending on the focus. This means that the generation system encodes no *a priori* assumptions about which domain concepts will be realized as which syntactic constituents.

Any lexical selection algorithm that handles such a wide range of constraints must deal with the issue of computational complexity. To ensure an efficient Lexical Chooser, we have developed a collection of techniques in the FUF programming environment to limit the cost of operations required during lexical choice. We have described in this paper lazy evaluation (with the `:wait` annotation) and dependency-directed backtracking (with the `:bk-class` annotation) and shown how it reduces the cost of lexical choice. With the help of these tools, we have implemented sophisticated lexical optimizations in FUF.

Our work results in an interpreter for an efficient Lexical Chooser, which allows handling of a wide variety of interacting constraints. In future work, we plan to investigate the construction of domain independent lexicon modules and methods for organizing large scale lexica, topics that we did not address in this work. This will require extending our work to provide tools for representing content of the lexicon in addition to the tools for manipulating and representing content that we have detailed here.

Acknowledgments
This research was partially supported by a joint grant from the Office of Naval Research and the Advanced Research Projects Agency under contract N00014-89-J-1782, by National Science Foundation Grants IRT-84-51438 and GER-90-2406, and by the New York State Science and Technology Foundation under this auspices of the Columbia University CAT in High Performance Computing and Communications in Health Care, a New York State Center for Advanced Technology. We would like to thank the anonymous reviewers for their helpful comments.

References

Abella, A. 1994. From pictures to words: generating locative descriptions of objects in an image. In *Proceedings of the Image Understanding Workshop*, Monterey, CA, November.

Anscombre, J.C. and O. Ducrot. 1983. *L'argumentation dans la langue*. Philosophie et langage. Pierre Mardaga, Bruxelles.

Appelt, D. 1983. Telegram: a grammar formalism for language planning. In *Proceedings of the 8th International Joint Conference on Artificial Intelligence*. IJCAI.

Bateman, J.A., R.T. Kasper, J.D. Moore, and R.A. Whitney. 1990. A general organization of knowledge for natural language processing: the penman upper-model. Technical report, ISI, Marina del Rey, CA.

Bourbeau, L., D. Carcagno, E. Goldberg, R. Kittredge, and A. Polguère. 1990. Bilingual generation of weather forecasts in an operations environment. In *Proceedings of the 13th International Conference on Computational Linguistics*, Helsinki University, Finland. COLING.

Boyer, M. and G. Lapalme. 1985. Generating paraphrases from meaning-text semantic networks. *Computational Intelligence*, pages 103–117, August-November.

Carcagno, D. and L. Iordanskaja. 1993. 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.

Carpenter, B. 1992. *The Logic of Typed Feature Structures with Applications to Unification Grammars, Logic Programs and Constraint Resolution*, volume 32 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, Cambridge, UK.

Conklin, E. 1983. *Data-driven indelible planning of discourse generation using salience*. Ph.D. thesis, University of Massachusetts, Ahmerst, MA.

Cousot, P. and R. Cousot. 1977. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of the Fourth ACM Symposium on the Principles of Programming Languages*, pages 238–252, Los Angeles.

Dalal, M., S.K. Feiner, K.R. McKeown, D. Jordan, B. Allen, and Y. alSafadi. 1996. Magic: An experimental system for generating multimedia briefings about post-bypass patient status. In *Proceedings of the American Medical Informatics Association*, Washington, D.C., October.

Dale, R. 1992. *Generating Referring Expressions*. ACL-MIT Press Series in Natural Language Processing, Cambridge, Ma.

Danlos, L. 1986. *The linguistic basis of text generation*. Studies in Natural Language Processing. Cambridge University Press.

de Kleer, J., J. Doyle, G.L. Steele, and G.J. Sussman. 1979. Explicit control of reasoning. In Winston P.J. and R.H. Brown, editors, *Artificial Intelligence: an MIT Perspective*. MIT Press, pages 93–116.

Elhadad, M. 1993a. Fuf: The universal unifier - user manual, version 5.2. Technical Report CUCS-038-91, Columbia University.

Elhadad, M. 1993b. Generating argumentative judgment determiners. In *Proceedings of the 11th National Conference on Artificial Intelligence*, pages 344–349. AAAI.

Elhadad, M. 1993c. *Using argumentation to control lexical choice: a unification-based implementation*. Ph.D. thesis, Computer Science Department, Columbia University.

Elhadad, M. 1995. Using argumentation in text generation. *Journal of Pragmatics*, 24:189–220.

Elhadad, M. 1996. Lexical choice for complex noun phrases. *Machine Translation*.

Elhadad, M. and J. Robin. 1992. Controlling content realization with functional unification grammars. In R. Dale, H. Hovy, D. Roesner, and O. Stock, editors, *Aspects of automated natural language generation*. Springer Verlag, pages 89–104.

Elhadad, M. and J. Robin. 1996. An overview of surge: a reusable comprehensive syntactic realization component. Technical Report 96-03, Ben Gurion University, Dept of Computer Science, Beer Sheva, Israel, April.

Elhadad, M., D. Seligmann, S. Feiner, and K. McKeown. 1989. A common intention description language for interactive multi-media systems. In *A New Generation of Intelligent Interfaces: Proceedings of IJCAI89 Workshop on Intelligent Interfaces*, pages 46–52, Detroit, MI, August 22.

Fawcett, R.P. 1987. 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.

Feiner, S. and K. McKeown. 1990. Generating coordinated multimedia explanations. In *Proceedings of the IEEE Conference on AI Applications*, Santa Barbara, CA, March.

Goldman, N. 1975. Conceptual generation. In Roger Schank, editor, *Conceptual Information Processing*. North-Holland, Amsterdam, pages 289–374.

group, The Penman NLG. 1989. The penman user guide. Technical report, Information Science Institute, Marina Del Rey, CA.

Halliday, M.A.K. 1985. *An introduction to functional grammar*. Edward Arnold, London.

Harbusch, K. 1994. Towards an integrated generation approach with tree-adjoining grammars. *Computational Intelligence*, 10(4):579–590.

Hovy, E. 1988. *Generating natural language under pragmatic constraints*. L. Erlbaum Associates, Hillsdale, N.J.

Iordanskaja, L., M. Kim, R. Kittredge, B. Lavoie, and A. Polguère. 1994. Generation of extended bilingual statistical reports. In *Proceedings of the 15th International Conference on Computational Linguistics*. COLING.

Jacobs, P. 1985. Phred: a generator for natural language interfaces. *Computational Linguistics*, 11(4):219–242.

Jacobs, P.S. 1987. Knowledge-intensive natural language generation. *Artificial Intelligence*, 33:325–378.

Kay, M. 1979. Functional grammar. In *Proceedings of the 5th Annual Meeting of the Berkeley Linguistic Society*.

Kukich, K. 1983a. The design of a knowledge-based report generator. In *Proceedings of the 21st Conference of the ACL*. ACL.

Kukich, K. 1983b. *Knowledge-based report generation: a knowledge engineering approach to natural language report generation*. Ph.D. thesis, University of Pittsburgh.

Kukich, K., K. McKeown, J. Shaw, J. Robin, N. Morgan, and J. Phillips. 1994. 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.

Lester, J.C. 1994. *Generating natural language explanations from large-scale knowledge bases*. Ph.D. thesis, Computer Science Department, Universtity of Texas at Austin, New York, NY.

Levi, J. 1978. *The syntax and semantics of complex nominals*. Academic Press.

Levin, B. 1993. *English verb classes and alternations: a preliminary investigation*. University of Chicago Press.

Lyons, J. 1977. *Semantics*. Cambridge University Press.

Mann, W.C. and C.M. Matthiessen. 1983. Nigel: a systemic grammar for text generation. Technical Report ISI/RR-83-105, ISI, Marina del Rey, CA.

Matthiessen, C.M. 1991. Lexicogrammatical choice in text generation. In C. Paris, W. Swartout, and W.C. Mann, editors, *Natural Language Generation in Artificial Intelligence and Computational Linguistics*. Kluwer Academic Publishers, Boston.

McCoy, K.F. and J. Cheng. 1991. Focus of attention: constraining what can be said next. In C. Paris, W. Swartout, and W.C. Mann, editors, *Natural Language Generation in Artificial Intelligence and Computational Linguistics*. Kluwer Academic Publishers, pages 103–124.

McDonald, D. 1981. Language production: the source of the dictionary. In *Proceedings of the 19th Annual Meeting of the Association for Computational Linguistics*. ACL.

McDonald, D. 1983. Description directed control: its implications for natural language generation. In Karen Sparck-Jones Barbara Grosz and Bonnie-Lynn Webber, editors, *Readings in Natural Language Processing.* Morgan Kaufmann Publishers.

McKeown, K. 1985. *Using Discourse Strategies and Focus Constraints to Generate Natural Language Text.* Studies in Natural Language Processing. Cambridge University Press.

McKeown, K. and M. Elhadad. 1990. A contrastive evaluation of functional unification grammar for surface language generators: A case study in choice of connectives. In *Natural Language Generation in Artificial Intelligence and Computational Linguistics.* Kluwer Academic Publishers, Boston. (also as Columbia technical report CUCS-407-88).

McKeown, K., K. Kukich, and J. Shaw. 1994. Practical issues in automatic documentation generation. In *Proceedings of the ACL Applied Natural Language Conference*, Stuttgart, Germany, October.

McKeown, K. and D. Radev. 1995. Generating summaries of multiple news articles. In *Proceedings of SIGIR*, Seattle, Wash., July.

McKeown, K., J. Robin, and K. Kukich. 1995. Generating concise natural language summaries. *Information Processing and Management, special issue on summarization*, 31(5):703–733, September.

McKeown, K., J. Robin, and M. Tanenblatt. 1993. 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.

McKeown, K. R., M. Elhadad, Y. Fukumoto, J.G. Lim, C. Lombardi, J. Robin, and F.A. Smadja. 1990. Text generation in comet. In R. Dale, C.S. Mellish, and M. Zock, editors, *Current Research in Natural Language Generation.* Academic Press, pages 103–140.

McKeown, K. R., S. Feiner, J. Robin, Seligmann D.D., and M. Tanenblatt. 1992. Generating cross-references for multimedia explanation. In *Proceedings of the 10th Annual Conference on Artificial Intelligence*, pages 9–16. AAAI.

Mel'cuk, I.A. and N.V. Pertsov. 1987. *Surface-syntax of English, a formal model in the Meaning-Text Theory.* Benjamins, Amsterdam/Philadelphia.

Meteer, M.W. 1990. *The generation gap: the problem of expressibility in text planning.* Ph.D. thesis, University of Massachussets at Ahmerst. Also available as BBN technical report No. 7347.

Meteer, M.W., D.D. McDonald, S.D. Anderson, D. Forster, L.S. Gay, A.K. Huettner, and P. Sibun. 1987. Mumble-86: design and implementation. Technical Report COINS 87-87, University of Massachussets at Amherst, Ahmerst, Ma.

Nogier, J.F. 1990. *Un systeme de production de language fonde sur le modele des graphes conceptuels.* Ph.D. thesis, Universite de Paris VII.

Paris, C. L. 1987. *The use of explicit user models in text generation: tailoring to a user's level of expertise.* Ph.D. thesis, Columbia University. Also available as technical report CUCS-309-87.

Passoneau, R., K. Kukich, J. Robin, V. Hatzivassiloglou, L. Lefkowitz, and H. Jing. 1996. Generating summaries of workflow diagrams. In *Proceedings of the International Conference on Natural Language Processing and Industrial Applications (NLP-IA'96)*, Moncton, New Brunswick, Canada.

Polguère, A. 1990. *Structuration et mise en jeu procédurale d'un modele linguistique déclaratif dans un cadre de génération de texte.* Ph.D. thesis, Université de Montreal, Quebec, Canada.

Pollard, C. and I. A. Sag. 1994. *Head Driven Phrase Structure Grammar.* University of Chicago Press, Chicago.

Pustejovski, J. and B. Boguraev. 1993. Lexical knowledge representation and natural language processing. *Artificial Intelligence*, 63(1-2):193–223.

Quirk, R., S. Greenbaum, G. Leech, and J. Svartvik. 1985. *A comprehensive grammar of the English language.* Longman.

Reiter, E.B. 1991. A new model for lexical choice for open-class words. *Computational Intelligence*, December.

Reiter, E.B. 1994. 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.

Robin, J. 1990. Lexical choice in natural language generation. Technical Report CUCS-040-90, Columbia University.

Robin, J. 1994a. Automatic generation and revision of natural language summaries providing historical background. In *Proceedings of the 11th Brazilian Symposium on Artificial Intelligence*, Fortaleza, CE, October.

Robin, J. 1994b. Revision-based generation of natural language summaries providing historical background: corpus-based analysis, design, implementation and evaluation. Technical Report CU-CS-034-94, Computer Science Department, Columbia Universtity, New York, NY. PhD. Thesis.

Robin, J. and K. McKeown. 1993. Corpus analysis for revision-based generation of complex sentences. In *Proceedings of the 11th National Conference on Artificial Intelligence*, pages 365–372. AAAI.

Robin, J. and K. McKeown. 1996. Empirically designing and evaluating a new revision-based model for summary generation. *Artificial Intelligence*, 85, August. Special Issue on Empirical Methods.

Rubinoff, R. 1992. *Negotiation, feedback and perspective within natural language generation.* Ph.D. thesis, Computer Science Department, University of Pennsylvania.

Shieber, S. 1992. *Constraint-based grammar formalism: parsing and type inference for natural and computer languages.* MIT Press, Cambridge, MA.

Shieber, S.M. and Y. Shabes. 1991. Generation and synchronous tree-adjoining grammars. *Computational Intelligence*, 7(4):220–228, December.

Shieber, S.M., G. Van Noord, R.M. Moore, and Pereira F.C.P. 1990. Semantic head-driven generation. *Computational Linguistics*, 16(1):30–42.

Smadja, F. 1991. *Retrieving collocational knowledge from textual corpora. an application: language generation.* Ph.D. thesis, Computer Science Department, Columbia University.

Smadja, F.A. and K. McKeown. 1991. Using collocations for language generation. *Computational Intelligence*, 7(4):229–239, December.

Strzalkowski, T. (Ed.). 1994. *Reversible grammars in natural language processing.* Kluwer Academic Publishers, Boston.

Swartout, W. 1983. The gist behavior explainer. In *Proceedings of the National Conference on Artificial Intelligence*, pages 402–407, Washington D.C. AAAI.

Talmy, L. 1976. Semantic causative types. In M. Shibatani, editor, *The grammar of causative constructions*, volume 6 of *Syntax and semantics.* Academic Press, London.

Talmy, L. 1983. How language structures space. In H.L. Pick and L.P. Acredolo, editors, *Spatial orientation: theory, research and application.* Plenum Press, New York/London.

Talmy, L. 1985. 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.

Van Noord, G. 1990. An overview of head-driven bottom-up generation. In R. Dale, C. Mellish, and M. Zock, editors, *Current Research in Natural Language Generation.* Academic Press, pages 141–166.

Winograd, T. 1983. *Language as a cognitive process.* Addison-Wesley.

Yang, G., K. McCoy, and K. Vijay-Shanker. 1991. From functional specification to syntactic structure: systemic grammar and tree adjoining grammars. *Computational Intelligence*, 7(4), December.

Zock, M. 1988. 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.