Controlling Content Realization with Functional Unification Grammars*

Michael Elhadad and Jacques Robin

Department of Computer Science
Columbia University
New York, NY 10027

Abstract

Standard Functional Unification Grammars (FUGs) provide a structurally guided top-down control regime for sentence generation. When using FUGs to perform content realization as a whole, including lexical choice, this regime is no longer appropriate for two reasons: (1) the unification of non-lexicalized semantic input with an integrated lexico-grammar requires mapping "floating" semantic elements which can trigger extensive backtracking and (2) lexical choice requires accessing external constraint sources on demand to preserve the modularity between conceptual and linguistic knowledge.

We introduce two control tools that we have implemented for FUGs to address these limitations: `bk-class`, a form of dependency-directed backtracking to efficiently process "floating" constraints and `external`, a co-routine mechanism allowing a FUG to cooperate with external constraint sources during unification. We show how these tools complement the top-down regime of FUGs to control the whole content realization process.

1 Introduction

Unification-based formalisms (Shieber 1986) and Functional Unification Grammars (FUGs) in particular, have proved popular in text generation. In previous work (Kay 1979, McKeown 1985, Appelt 1985, Paris 1987), the input to FUGs was a fully lexicalized specification. In recent work, however, the functionality of FUGs has been extended to encompass all of content realization, including lexical choice (McKeown et al 1990, Elhadad 1991b, Smadja 1991b, Robin 1992). In this framework, content realization is viewed as the process of unifying a purely semantic input with an integrated *lexico*-grammar, in the systemic sense (Matthiessen 1991).

When a non-lexicalized semantic structure is accepted as input, the FUG has the burden of mapping this structure onto a syntactic structure. This additional task increases the complexity of the unification process because semantic and syntactic structures are not isomorphic:

* Reprinted from *Aspects of Automated Natural Language Generation*, R. Dale, E. Hovy, D. Rösner and O. Stock editors, Springer Verlag, 1992, pp.89-104.

- Several elements in the semantic structure can be realized by a single syntactic constituent (*e.g.,* in a sports report, the verbal pattern *"X stunned Y"* expresses both that *X* is the winner of a game and that *Y* was the favorite).
- The same semantic element can be realized by linguistic constituents at different linguistic ranks (*e.g.,* the surprise expressed by *"X stunned Y"* can alternatively be conveyed by an adverbial phrase, as in *"against all odds, X defeated Y"* or an adjectival phrase *"X defeated the highly favored Y"*).

In FUGs, unification was traditionally controlled by a top-down regime guided by the input structure. When accepting a semantic input containing "floating" elements that can be realized by constituents at different levels of the syntactic tree, such a top-down regime can trigger expensive backtracking.

Moreover, lexical choice is constrained by many different factors: encyclopedic, interlexical, grammatical, discursive and interpersonal (Robin 1990). All of these constraints cannot be integrated in a single FUG. In a modular architecture, they must be provided by independent knowledge sources. To perform lexical choice, the FUG must interact with these various knowledge sources during unification.

To address these new needs, we propose to integrate explicit control annotations within FUGs. Specifically, we present two control tools that we have implemented in FUF (Functional Unification Formalism), our extended version of FUGs (Elhadad 1991a, 1990):

- `bk-class`, a form of dependency directed backtracking, used to handle floating constraints efficiently.
- `external`, a co-routine interface used to query external constraint sources from the FUG.

In this paper, we first describe the standard control regime used in FUF. We then introduce the `bk-class` construct and we quantitatively evaluate how it reduces backtracking for the realization of floating constraints. We then present the `external` tool and illustrate its use to query a domain knowledge base at unification time.

2 Standard Control in FUF

FUG relies on the primitive operation of unification of Functional Descriptions (FDs) (Kay 1979). FDs are sets of pairs `(a v)`, called features, where `a` is an attribute and `v` is a value. A value is either : (1) an atom, (2) recursively an FD or (3) a path to another feature in the FD.[2] A given attribute `a` is allowed to appear at most once in a given FD. Two simple FDs are compatible if they do not include a contradictory

[2] Value paths are used to specify that two features share the same value. In FUF, a path is a list of embedded attributes surrounded by curly braces. This list can be prefixed by a number of carets, making the path *relative* to the level of embedding of the feature. An attribute is generally an atom. However, it can also be a path to allow for the specification of its value at various embedding levels in the recursive structure of an FD.

value for the same attribute.[3] When they are compatible, the unification of two FDs merges the features from both to produce a more specific FD, the *total FD*.

There are four constructs of FUF that are of importance in this paper: `alt`, `cset`, `any` and `given`. The `alt` keyword expresses disjunction in FUG. 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.

During sentence generation, unification is used to add linguistic information from a functional unification grammar (FUG) to a semantic input, both represented as FDs. Figure 1 shows the input to generate the sentence *"Robinson scored 32 points"* and a FUG (grammar G1) specifying the mapping from semantic categories to syntactic categories. When unifying this semantic input I1 with G1 the following operations are performed: FUF picks branch 2 of the `alt` and merges it with the input. During the merging, the features uppercased in Fig. 1 get added to the result.

The semantic input is a structured representation. It consists of a top-level predicate with embedded arguments. In the single unification shown in Fig. 1, however, only the top-level FD is enriched. The FDs embedded under `agent` and `medium` are not enriched. To properly refine the structured semantic input into a syntactic description we need to process these sub-FDs, by reaccessing the grammar at each level.

The way FUF proceeds at this point, is based on the notion of *constituent*: a constituent of a complex FD is a distinguished sub-FD. The special label `cset` (Constituent Set) identifies constituents. The value of `cset` is a list of attributes naming the constituents of the FD as shown in Fig. 1. Intuitively, constituents bring structure to functional descriptions.

To handle constituents, the complete unification procedure is:

1. Unify top-level input with grammar (single unification).
2. Identify constituents in result.
3. Recursively unify each constituent with the grammar.

Constituents therefore trigger recursion in FUGs. However, this description of the unification mechanism does not specify what control regime must be used to traverse the constituent structure. FUF implements the following regime: *top-down* and *breadth-first* traversal of the constituent structure. At each level of the structure, constituents are processed in the order they are declared in the `cset`. So in our example FD, the constituent structure is processed as follows: top-level first, then `agent`, then `medium`. The resulting FD at the end of the process is shown at the bottom of Fig. 1.

The two remaining FUF constructs we need to discuss in this section are the meta-variables `given` and `any`.

A well-known problem with top-down control regimes is their handling of left-recursive rules. For example, the grammar for NPs specifies that the determiner of

[3] There is no notion of variable in functional unification. Failure can only occur at the leaf of a feature structure when trying to unify two different atoms for the same attribute.

Semantic input I1:

```
((sem-cat action) (concept c-score) (tense past)  ;; Predicate
 (agent ((sem-cat individual) (concept c-player)  ;; Argument 1
         (name Robinson)))
 (medium ((sem-cat set) (concept c-stat)          ;; Argument 2
          (unit c-point) (cardinal 32))))
```

Grammar G1:

```
( ...
 (alt
  (((sem-cat individual) (synt-cat proper-name))              ;; Branch 1
   ((sem-cat action) (synt-cat clause) (cset (agent medium))) ;; Branch 2
   ((sem-cat set) (synt-cat np))))                            ;; Branch 3
  ... )
```

Total FD after a single (top-level) unification:

```
((sem-cat action) (concept c-score) (tense past)
 (SYNT-CAT CLAUSE) (CSET (AGENT MEDIUM))
 (agent ((sem-cat individual) (concept c-player) (name Robinson)))
 (medium ((sem-cat set) (concept c-stat) (unit c-point) (cardinal 32))))
```

Total FD after recursive constituent unification:

```
((sem-cat action) (concept score) (tense past)
 (synt-cat clause) (cset (agent medium))
 (agent ((sem-cat individual) (concept c-player) (name Robinson)
         (SYNT-CAT PROPER-NAME)))
 (medium ((sem-cat set) (concept c-point) (cardinal 32)
          (SYNT-CAT NP))))
```

Fig. 1. An example of unification

an NP can be a possessive NP. In a phrase-structure formalism, this is encoded by a left-recursive rule such as $np/NP \rightarrow det/NP, nbar/NP$ (Shieber et al 1989, p.10). With a top-down regime, such a rule can lead to non-termination. In FUF, this problem is avoided by the use of the special construct `given`. The `given` construct checks that a given feature is instantiated in the input FD *before* unification starts. By adding a pair `(possessor given)` in the NP sub-grammar, we can ensure that a possessive NP gets added in determiner position only when a `possessor` constituent is given in the semantic input. Thus, recursion is only triggered when necessary and will always terminate.

Another meta-variable, `any`, implements a powerful delaying mechanism. A feature `(x any)` constrains `x` to be instantiated with *some* value at the *end* of the unification. The unifier enforces this constraint as follows: if `x` is already instantiated in the input FD, then `any` is satisfied; if it is not yet instantiated, the constraint

is delayed and checked again at the end of the unification process. If at this point `x` is still not instantiated, the constraint fails and the unifier needs to backtrack. Therefore `any` is a meta-variable that triggers a delayed check.

In contrast to a procedural implementation of systemic grammars like NIGEL (Mann and Matthiessen 1983), where the control regime is driven by an *a priori* grammatical structure, FUF's control regime is driven by the structure of the semantic input FD. It is therefore similar in spirit to the semantic-head-driven algorithm presented by Shieber et al (1989). It also avoids the inefficiencies associated with a bottom-up control regime, *e.g.,* the backtracking introduced by choosing the case of an NP before knowing what syntactic role it will fill in a clause.

The control regime described thus far is the default regime followed by FUF. A more flexible control regime can be implemented by explicitly controlling the value of the `cset` feature in different parts of the grammar. In this paper, we do not attempt to cover the many control issues related to lexical choice. Instead, we focus on the special control devices we have implemented in FUF. A more complete description of various control regimes for lexical choice implemented in FUF can be found in (Elhadad 1992b) and (Robin 1992).

3 Bk-class and Floating Constraints

The task of the generator is to map from a semantic constituent structure to a syntactic one. This task is difficult because, in general, these structures are not isomorphic: *"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, p. 57). For example, in the basketball domain, the clause pattern *"X edged Y"* conveys two semantic elements: a semantic predicate - X won a game against Y - and a manner qualification - the game was close.[4]

Moreover, the same semantic element can be realized by syntactic elements at different linguistic ranks (*e.g.*, group, clause, sentence) For example, the low rating of a team can be conveyed by a variety of syntactic constituents:

- Adjective (at the noun-group rank): *The hapless Denver Nuggets beat the Boston Celtics 101-99.*
- Verb (at the verb-group rank): *The Denver Nuggets stunned the Boston Celtics 101-99.*
- Adverbial (at the clause rank): *The Denver Nuggets surprisingly beat the Boston Celtics 101-99.*

We call such semantic elements *floating constraints*. We distinguish them from *structural constraints* such as semantic predications or references. Structural constraints require the presence of syntactic constituents at a given linguistic rank in the output and thus guide the mapping process from the semantic structure to the syntactic structure. For example, when an input event structure is mapped to a

[4] 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 (Talmy 1976 and 1983).

clause, the semantic predicate `c-win` determines how the semantic roles are mapped onto syntactic complements: `winner` to `subject` and `loser` to `object`.

The top-down regime implemented in FUF handles structural constraints efficiently because backtracking is circumscribed to the unification of the grammar with a *single* input constituent. In contrast, the processing of floating constraints can be very inefficient because it can trigger non-local backtracking, cutting across linguistic ranks and requiring the re-unification of the grammar with *several* input constituents. To illustrate this problem, consider a system reporting on the results of a basketball game and an input containing the following three constraints:

- Semantic Predication: convey that the Denver Nuggets defeated the Boston Celtics by a 101-99 score.
- Manner Qualification: convey that the game was tight.
- Argumentative Orientation: convey the low rating of the Denver Nuggets.

For example, the above input configuration of constraints is correctly satisfied by the following sentence: *"The hapless Denver Nuggets edged the Boston Celtics 101-99."*

But all these different linguistic devices cannot be freely combined, as illustrated by the following examples:

1. *? The Denver Nuggets narrowly stunned the Boston Celtics 101-99.*
2. *? The Denver Nuggets surprisingly nipped the Boston Celtics 101-99.*
3. *? Against all odds, the Denver Nuggets narrowly beat the Boston Celtics 101-99.*

In sentence (1), it is not clear which semantic aspect of the verb *"stunned"* is modified by *"narrowly"*: the expression of the game result or its unexpectedness. Similarly in (2), the modification of *"nipped"* by *"surprisingly"* is ambiguous. In (3), the scope of *"against all odds"* is ambiguous: it could be either *"narrowly"* - in which case the Nuggets are presented as highly rated - or *"beat"* - in which case the Nuggets are presented as lowly rated.

The input FD shown at the top of Fig. 2 encodes the three constraints we want to satisfy. The central part of Fig. 2 shows a fragment of a lexicon specifying the mapping between concepts and lexical items. The fragment shows how different verbs impose constraints on the features `AO` or `manner`, or no constraint for "neutral" verbs. The branch order in the `win-lex` alt enforces the stylistic preference for semantically rich verbs over neutral verbs with adverbials.[5] In addition, to avoid generating adverbials with ambiguous scope, the grammar enforces that (1) clauses contain a single adverb and (2) only neutral verbs are used in combination with adverbials.

Consider the realization of the semantic input at the top of Fig. 2 with the grammar at the bottom of this figure. FUF's top-down regime allows it to map the structural constraints to syntactic constituents right away: first the semantic predicate to a verb-group, and then the roles `winner`, `loser` and `score` to `subject`, `object`

[5] FUF tries the branches of an `alt` construct in order. When no order is preferable, the construct `ralt` (standing for Random Alternation) whose branches are tried at random is used instead.

A semantic input expressing three constraints:

```
(;; Semantic predication
 (sem-cat action) (concept c-win) (token t-win-666) (tense past)
 (winner ((sem-cat individual) (concept c-team) (name Nuggets)))
 (loser ((sem-cat individual) (concept c-team) (name Celtics)))
 (score ((sem-cat quantity) (concept c-game-score)
         (winner-score 101) (loser-score 99)))
 ;; Manner constraint
 (manner ((sem-cat quality) (concept c-tight)))
 ;; Argumentative constraint
 (AO ((sem-cat scale) (concept c-rating)
      (carrier {winner}) (orientation -))))
```

Choice of verb in the lexico-grammar:

```
(...
 ((sem-cat action)
  (alt (index on concept)
    ;; Map the concept game-result to a verb
    (((concept win)
      (alt win-lex (:bk-class (AO manner))
        (;; The winner's rating is poor
         (({AO} ((concept c-rating) (carrier {winner})
                 (orientation -) (conveyed yes)))
          (lex ((alt ("stun" "surprise"))))
          ({adverb} none))

         ;; The victory is narrow
         (({manner} ((concept c-tight) (conveyed yes)))
          (lex ((alt ("edge" "nip"))))
          ({adverb} none))

         ;; Default neutral verbs
         ((lex ((alt ("beat" "defeat" "down"))))))))
       ... ))))
```

Floating constraints mapping in the lexico-grammar:

```
( ...
 (verb ((alt ...))) ...
 (AO ((alt ...))) ...
 (manner ((alt manner-adverbial (:bk-class manner)
           (;; Can be realized by other means - delay
            ((manner-conveyed any))

            ;; Map manner to an adverbial adjunct
            ;; and mark that manner has been realized
            (({adverb} ((synt-cat adverb) (concept {^ ^ concept})))
             (manner-conveyed adverb))))))
  ... )
```

Fig. 2. Handling floating constraints with `bk-class`

and `adjunct` respectively. In contrast, the mapping of the "floating" constraints `AO` and `manner` must be delayed.

Figure 2 illustrates this delaying mechanism by showing how the `manner` input constraint is handled by the grammar. The feature `manner-conveyed` is used to record the syntactic category of the constituent realizing the manner constraint. It remains `nil` as long as the constraint is not conveyed. In the first branch of the `manner-adverbial` alt, we first check whether the manner constraint has already been handled by some other constituent. This check is implemented by the feature `(manner-conveyed any)`. This first branch delays the decision to use an adverb with the `any` construct. This gives other devices a chance to express the manner constraint. However if no other linguistic device can be found that satisfies the manner constraint, the grammar resorts to using an adverbial adjunct, by choosing the second branch of the alt. The feature `(manner-conveyed any)` therefore prevents the generation of semantically incomplete sentences like *"The Denver Nuggets stunned the Boston Celtics 101-99."* The argumentative constraint is similarly handled with a feature `ao-conveyed`.

Let us now consider how the manner and argumentation constraints interact. In a top-down regime, the verb-group is first processed and the concept `c-win` is lexicalized. FUF is now traversing the lexicon fragment in the middle of Fig. 2 and first chooses the verb "stun" which satisfies both the semantic predication and the argumentative constraint. It then maps the semantic constituents to syntactic functions and proceeds to the argumentative constraint. This constraint is already satisfied by the verb, so no modifier needs to be introduced.

At this point FUF attempts to take into account the manner constraint. It first delays the use of an adverb with the `any` construct and completes the traversal of the constituents top-down. It eventually checks the `any` construct and finds that the manner constraint has not been satisfied. Backtracking is triggered. Consider at this point the state of the backtracking-point stack: the whole grammar has been traversed, all the subconstituents processed. Basically, all potential backtracking points are on the stack. If FUF blindly backtracks, search is maximized. Since we cannot know *a priori* where in the syntactic structure the floating manner constraint will fit, the decision whether to use an adverb must be delayed until the end of the traversal. There is therefore no way to detect failure before this point.

To avoid the cost of a blind backtracking, we introduce the `bk-class` construct. It implements 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 certain location in the total FD. In our example, we have to backtrack because an equation requires that the value of the feature {`manner manner-conveyed`} be instantiated, but the actual feature is not. The path {`manner manner-conveyed`} defines the *address of the failure.*[6]

The idea is that the location of a failure can be used to identify the only decision points in the backtracking stack that could have caused it. This identification requires

[6] In an FD, each embedded feature can be viewed as an equation between the path leading to the feature in the total FD and the feature value.

additional knowledge that must be declared in the FUG. More precisely, we first allow the FUG writer to declare certain paths 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`.

For example, the statement: `(define-bk-class manner {manner manner-conveyed})` specifies that the path `{manner manner-conveyed}` is of class manner. In addition, we tag in the FUG all `alts` that have an influence on the handling of the manner constraint with a declaration `(:bk-class manner)` as shown in Fig. 2.

When the unifier fails at a location of class `manner`, it *directly* backtracks to the last choice point of class `manner`, ignoring all intermediate decisions. In our example, when the `any` constraint fails, we directly backtrack to the manner choice point in the grammar (bottom of Fig. 2). If this last option fails again, we backtrack up to the choice of verb in the lexicon (middle of Fig. 2). We therefore use the knowledge that *only* the verb or the adverb can satisfy the manner constraint in a clause to drastically reduce the search space. But, this knowledge is *locally* expressed at each relevant choice point, retaining the possibility of independently expressing each constraint in the FUG.

In general, the determination of the address of failure is more complex and it is necessary to distinguish between *initial failures* and *derived failures*. An initial failure always occurs at a leaf of the total FD, when trying to unify two incompatible atoms. Failures however can also propagate up the structure of the total FD. For example, when unifying `((a ((b 1))))` with `((a ((b 2))))` the original address of failure is the path `{a b}`. When the unifier backtracks, it also triggers a failure at address `{a}`, which is not a leaf. This type of failure is called a derived failure. In the implementation of `bk-class`, FUF ignores derived failures and directly backtracks to the first choice point whose `bk-class` matches the last initial failure.

For the example of Fig. 2, we have measured the number of backtracking points required to generate different clauses conveying the same core content. Table 1 summarizes these measurements.[7]

Table 1. Measuring the effect of `bk-class`

		Backtracking points	
Input	Output	w/o `bk-class`	w/ `bk-class`
No floating constraints	*The DN beat the BC*	110	110
Manner in the verb	*The DN edged the BC*	110	110
Manner as adverbial	*The DN narrowly beat the BC*	>10000	214
AO in the verb	*The DN stunned the BC*	112	112
AO as adjective	*The hapless DN beat the BC*	1,623	239
AO as adverbial	*The DN surprisingly beat the BC*	>100,000	277
AO & manner together	*The hapless DN edged the BC*	1,178	238

The number of backtracking points required to generate each example clause is

[7] In this table, *DN* abbreviates *Denver Nuggets* and *BC* abbreviates *Boston Celtics*.

listed with and without `bk-class`. The numbers for the first clause, which does not include any floating constraints, give an indication of the size of the grammar. It can be interpreted as the number of decisions the grammar makes to generate a basic clause for which practically no backtracking is required. It roughly corresponds to the number of unretracted decisions made by the grammar. It is the optimal number of backtracking points that a search control regime can obtain for the given input with this grammar. 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 grammar 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. In general, it would be desirable to delay the attachment until it is proven compatible with the other constraints. In FUF, an explicit annotation called `wait` implements such a delaying mechanism. It is similar to Naish's implementation for Prolog (Naish 1985). In FUF, a `wait` annotation freezes the choice of a branch in a disjunction until values for a given set of paths in the total FD are available.

While backtracking can be minimized by the use of `wait`, it cannot be avoided entirely. When FUF's input contains several mutually dependent floating constraints, they are all delayed in a deadlock situation. To break the deadlock, FUF selects one of the constraints arbitrarily. This non-deterministic choice can lead to backtracking. In this case, a combination of `bk-class` and `wait` is necessary to minimize backtracking. We are currently evaluating the efficiency gains of this combination of control tools over the use of `bk-class` alone.

The `bk-class` mechanism improves FUGs' efficiency while preserving their desirable properties - declarativeness and bidirectional constraint satisfaction. It can be declaratively read as a statement of dependency between a decision in the grammar and a class of constraints in the input. Using `bk-class`, however, is not always easy for the grammar writer since it requires thinking about the control strategy of the unifier - the same drawback as for Prolog's `cut` mechanism. But `bk-class` annotations are optional, and can be added only when needed to optimize a grammar.

4 External and Modularity

Content realization consists of mapping a semantic input structure onto a syntactic tree. In addition to the constraints present in the input, this process is constrained by a heterogeneous set of factors. Such factors are surveyed in (Matthiessen 1991) and (Robin 1990). They include:

- grammar rules
- a conceptual lexicon specifying the mapping between domain concepts and lexical items
- a grammatical dictionary providing the special grammatical properties of lexical items
- a collocation dictionary providing the restrictions on lexical co-occurrences
- a discourse model keeping track of the structure of the text as it is generated
- a domain knowledge base representing the encyclopedic context of generation
- a user-model representing the interpersonal context of generation

These sources vary along several dimensions:

- *Structure*: the grammar rules and the conceptual lexicon express *structural* constraints. They specify a transformation from one regular structure to another. Other sources like the collocation dictionary express inherently non-structural constraints (Halliday 1976, p. 73).
- *Portability*: the grammar rules, the grammatical dictionary (Cumming 1986) and to some extent the collocation dictionary (Smadja 1991a) are domain-independent. The other sources are highly domain-dependent.
- *Dynamism*: the discourse model is inherently dynamic, changing from one sentence to the next. In some applications (Dale 1988), this is also the case for the domain model and the user-model. The other sources are static.

How can these knowledge sources be combined?

One approach would be to integrate all these constraints into a single FUG. In addition to being non-modular and thus hindering portability, this approach is impractical for dynamic constraints: being a monotonic process, unification is inadequate to update dynamic models as generation unfolds.

A modular architecture is therefore preferable. The structural constraint sources - conceptual lexicon and grammar rules - can readily be implemented as a FUG as they are well handled by FUF's top-down regime. During unification, this backbone FUG needs to communicate with the other sources when necessary. What we need is a mechanism allowing constraints that lie outside of both the input FD and the FUG to be taken into account at any point during the unification process.

We introduce the `external` construct to address this need. When FUF encounters a feature of the form `(a #(external F))` it performs the following operation:

1. Unification is suspended.
2. The external function `F` - a LISP function returning a sub-FD - is evaluated.
3. The value returned by `F` becomes the new value of the attribute `a` in the total FD.
4. Unification resumes where it was suspended with the updated total FD.

Therefore `External` allows the dynamic expansion of a FUG at unification-time.

To illustrate the use of the `external` construct in FUF, consider again the task of generating the sentence: *"The hapless Denver Nuggets edged the Celtics 101-99."* As explained in Sect. 3, the verb *"to edge"* in this sentence not only realizes the predicate element of the semantic input but also the manner constraint. This

floating constraint provides a qualitative evaluation of the basketball game reported by the sentence. Such a qualitative evaluation does not depend only on the final score of the game[8] but on other quantitative factors as well, such as the number of lead changes in the final minutes or the largest lead by either team at any point during the game. Several such quantitative facts about a game are abstracted and conflated with the semantic predicate by verbs like *"to edge"*, *"to hammer"*, *"to outlast"* or *"to rally past"*. Choosing among these verbs requires deciding which combination of quantitative facts is abstracted by the manner connotation of each verb.

In Sect. 3, we assumed that this decision was performed by the content planner building the FUG semantic input as illustrated by the presence of the `(concept c-tight)` feature in the input of Fig. 2. Performing such a mapping, however, requires knowledge of the existing lexical resources available in a given sublanguage (Kittredge and Lehrberger 1983). The fact that a given combination of quantitative data about a basketball game can be compactly expressed in English by describing the game as *"tight"* is *linguistic knowledge*. It should therefore be located in the lexicon portion of the FUG. Using `external` allows FUF to enforce this separation between linguistic and conceptual knowledge. In this case, the semantic input to the FUG no longer needs to provide a pre-linguistic specification of the manner. It needs only indicate that one of the sentence's communicative goals is to express the manner. Instead of the feature: `(manner ((sem-cat quality) (concept c-tight)))`, the semantic input just contains the feature: `(manner any)`. The lexico-grammar is now in charge of choosing a lexical item to appropriately qualify the game. To perform this choice, it must access the description of the game in the encyclopedic knowledge base. Figure 3 shows a fragment of a lexicon where the `external` construct implements an example of such query.

Unification of the semantic input with the `win-lex alt` of this lexicon fragment triggers calls to external functions. Each of these functions queries the knowledge base for quantitative data and returns an FD containing corresponding qualitative features. For example, the function `get-lead-changes` shown in Fig. 3 enriches the total FD with the feature `lead-changes`. These external functions connect the FUG with the knowledge base. Within the body an external function, one can access any feature in the total FD by specifying its path prefixed by @. In Fig. 3, note how this notation is used in the `get-lead-changes` function to retrieve information from the knowledge base about the particular token given in the semantic input.[9]

After the external functions return, the features added to the total FD are used for choosing a verb conveying the manner connotation appropriate to this particular game. For example, the verb *"to edge"* is preferred when a combination of features signals that (1) there was no overtime (2) no team built a big lead and (3) there were many lead changes. As we have identified about 100 different verbs in the basketball sublanguage to express the victory of a team, many features are required to discriminate between them. However, in a given situation, only a few features will actually be needed. Having the content planner systematically retrieve all the

[8] In which case it would be redundant with the quantitative expression of the score in the sentence.

[9] Recall that the semantic input is part of the total FD at any point during unification.

An external query function:

```
(defun get-lead-changes ()
  (let ((lead-change-num (get-role-value (get-token @{token})
                                         'lead-change-num)))
    (cond ((> lead-change-num 15) `((lead-changes numerous)))
          ((and (> lead-change-num 5) (> 15 lead-change-num))
           `((lead-changes average)))
          ((> 5 lead-change-num) `((lead-changes few))))))
```

Backbone lexico-grammar with `external` *constructs*:

```
(...
 ((sem-cat action)
  (concept c-win)
  (alt win-lex (:bk-class (A0 manner))
    ((({manner} any)

      ;; Knowledge base query for information discriminating
      ;; among manner-conveying verbs.
      ({manner} ((overtime #(external #'get-overtime))
                 (biggest-lead #(external #'get-biggest-lead))
                 (lead-changes #(external #'get-lead-changes))
                 ...))
      (alt win-and-manner-lex

        (;; The victory was obtained in overtime after many lead changes
         (({manner} ((overtime yes) (lead-changes numerous)))
          (lex "outlast"))

         ;; The victory was close and obtained in regulation
         (({manner} ((overtime no) (biggest-lead small)
                     (lead-changes numerous)))
          (alt (((lex "edge") (lex "nip")))))
         ...))))))))
```

Fig. 3. Accessing an external knowledge source from the lexicon

knowledge base information necessary to discriminate between all the words of the lexicon would thus be computationally wasteful.

In addition, the set of features required to discriminate between words depends on the part of speech. For example, there are many fewer adverbs available to convey the manner connotation than there are verbs. Fewer features are therefore required to select an adverb than to select a verb. Requiring the content planner to provide the features for all classes of lexical choice in advance would therefore impair modularity between linguistic and conceptual knowledge.

To summarize, the `external` construct enhances FUF in the following ways:

- It provides a co-routine control structure to interact with external processes.
- It enforces an information-hiding principle between different knowledge sources.
- It is a way to fetch constraints lying outside the FUG *on demand*, only when needed by FUF to choose between alternatives.

These different points correspond to needs that have been identified in many generation systems. TELEGRAM (Appelt 1985) implemented a mechanism where a FUG and a content planner cooperated to generate referring expressions. The `external` construct is a generalization of this mechanism. With PAULINE, Hovy (1988) advocated interleaving pervasively content realization with content planning. With `external`, FUF can implement such an interleaving while benefiting from the declarative nature of FUGs. While traversing a systemic linguistic network, PENMAN (Mann 1983) accesses its environment by calling functions called *inquiries*. FUF's `external` functions provides a similar facility in the context of FUGs. Finally, DIOGENES, (Nirenburg and Nirenburg 1988) uses a blackboard to communicate with and control specialized modules working with their own separate knowledge sources. With `external`, a similar cooperation among specialized modules can be implemented in FUF with the total FD playing the role of the blackboard.

5 Conclusion

In this paper, we have addressed the issue of using FUGs to perform content realization as a whole, including lexical choice. When unifying a non-lexicalized semantic input with an integrated lexico-grammar, two new problems occur: (1) dealing with floating constraints and (2) accessing external knowledge sources. We have presented two control tools in FUGs to address these problems: `bk-class` and `external`.

To improve efficiency, `bk-class` implements a form of dependency-directed backtracking, taking advantage of the knowledge of what choice points in a grammar can influence the realization of a floating constraint. Naish (1985, p. 59) lists heuristics to improve efficiency in search, including "detect failure early" and "avoid failure." We have shown in Sect. 3 that there are good linguistic reasons why an early detection of failure for "floating" constraints is very difficult. In such cases, `bk-class` implements the heuristic of avoiding failure.

To achieve modularity in text generators, `external` implements a co-routine mechanism for communication between a FUG and other knowledge sources during unification. This mechanism generalizes approaches introduced in earlier work and addresses a criticism often expressed against FUGs.

Both `bk-class` and `external` augment the general semantic structure driven top-down regime of FUGs. They have been implemented and tested extensively in a wide variety of applications: COMET (McKeown et al 1990), a system that generates explanations in a multimedia setting, COOK (Smadja 1991b) a sentence-generation system that addresses the issue of collocations in stock market reports, ADVISOR (Elhadad 1991b, 1992a), a question-answering system that generates argumentative paragraphs and STREAK (Robin 1992) a system that generates information-packed report leads in the basketball domain.

Acknowledgments Many thanks to Kathy McKeown, David Kurlander, Becky Passoneau and Tony Weida. The research reported in this paper was partially supported by ONR grant N00014-89-J-1782, DARPA grant N00039-84-C-0165 and NSF grants IRT-84-51438 and IRI-90-24069.

References

D. Appelt. *Planning Natural Language Utterances.* Studies in Natural Language Processing. Cambridge University Press, 1985.

S. Cumming. The lexicon in text generation. In *Proceedings of the 1st Workshop on Automating the Lexicon*, Pisa, Italy, 1986.

R. Dale. *Generating referring expressions in a domain of objects and processes.* PhD thesis, University of Edinburgh, Scotland, 1988.

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

M. Elhadad. Types in functional unification grammars. In *Proceedings of the 28th Annual Meeting of the Association for Computational Linguistics*, Detroit, MI, 1990. ACL.

M. Elhadad. FUF: The universal unifier - user manual, version 5.0. Technical Report CUCS-038-91, Columbia University, 1991.

M. Elhadad. Generating adjectives to express the speaker's argumentative intent. In *Proceedings of the 9th Annual Conference on Artificial Intelligence.* AAAI, 1991.

M. Elhadad. Generating coherent argumentative paragraphs. Submitted to COLING'92, 1992.

M. Elhadad. *Using argumentation to control lexical choice: A functional unification-based approach.* PhD thesis, Computer Science Department, Columbia University, 1992.

M.A.K. Halliday. *System and function in language.* Oxford University Press, Oxford, 1976.

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

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

R. Kittredge and J. Lehrberger. *Sublanguages: studies of language in restricted semantic domains.* Walter DeGruyter, New York, 1983.

W.C. Mann. An overview of the PENMAN text generation system. Technical Report ISI/RR-83-114, ISI, Marina del Rey, CA, 1983.

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

C.M. Matthiessen. 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, 1991.

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

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

Lee Naish. *Negation and Control in Prolog*, volume 238 of *Lectures Notes in Computer Science.* Springer Verlag, 1985.

S. Nirenburg and I. Nirenbrug. A framework for lexical selection natural language generation. In *Proceedings of the 11th International Conference on Computational Linguistics*. COLING, 1988.

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

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

J. Robin. A revision-based architecture for reporting facts in their historical context. To appear in Horacek H. and M. Zock, editors, *Proceedings of the Third European Workshop on Language Generation*, 1992.

S. Shieber. *An introduction to Unification-Based Approaches to Grammar*, volume 4 of *CSLI Lecture Notes*. University of Chicago Press, Chicago, Il, 1986.

F. Smadja. Microcoding the lexicon with co-occurrence knowledge. In Uri Zernik, editor, *Lexical Acquisition: Using on-line resources to build a lexicon*. Lawrence Erlbaum, 1991.

F. Smadja. *Retrieving Collocational Knowledge from Textual Corpora. An Application: Language Generation*. PhD thesis, Computer Science Department, Columbia University, 1991.

S.M. Shieber, G. Van Noord, R.M. Moore, and Pereira F.C.P. A semantic head-driven generation algorithm for unification-based formalisms. In *Proceedings of the 27th ACL*, pages 7–17, Vancouver, British Columbia, Canada, 1989. ACL.

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

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

L. Talmy. Lexicalization patterns: semantic structure in lexical form. In T. Shopen, editor, *Grammatical categories and the lexicon*, volume 3 of *Language typology and syntactic description*. Cambridge University Press, 1985.

This article was processed using the LaTeX macro package with LMAMULT style