CONTROL IN FUNCTIONAL UNIFICATION GRAMMARS FOR TEXT GENERATION

Michael Elhadad
Jacques Robin

Technical Report
CUCS-020-91
Department of Computer Science
Columbia University
New York, NY 10027

Elhadad@cs.columbia.edu
Robin@cs.columbia.edu

Topic Area: Text generation, Functional Unification Grammars, Control.

Abstract

Standard Functional Unification Grammars (FUGs) provide a structurally guided top-down control regime for text generation that is not appropriate for handling non-structural and dynamic constraints. We introduce two control tools that we have implemented for FUGs to address these limitations: `bk-class`, a tool to limit search by using a form of dependency-directed backtracking and `external`, a co-routine mechanism allowing a FUG to cooperate with dynamic constraint sources. We show how these tools complement the top-down regime of FUGs to enhance lexical choice.

1 INTRODUCTION

Text generation is the process of choosing linguistic devices to satisfy various constraints. One of these constraints is the semantic structure representing the information to be conveyed. Mapping this structure to a syntactic tree is well handled by a top-down traversal of the semantic structure (as illustrated by the semantic-head-driven algorithm described in [21]). However, inherently non-structural sources of constraints, such as lexical collocations [11, 24] and pragmatic constraints [12] also affect generation. Due to lack of structural guidance, scheduling the application of these constraints is problematic. Moreover, different types of constraints are provided by different knowledge sources *e.g.*, knowledge base, lexicon, grammar, user-model. In a modular architecture, independent sources cannot know exactly what information they should provide before the process of combining constraints starts.

Standard Functional Unification Grammars (FUGs) are widely used in surface text generation [16, 1, 19, 17] and provide the regular top-down control most appropriate for structural constraints. However they do not satisfactorily address the problems of scheduling non-structural constraints and communication between several independent knowledge sources. We have extended FUGs in an implemented system called FUF [6, 7] and used this implementation extensively for many applications in text generation [2, 9, 23] and non-linguistic applications [10]. The standard control mechanism provided by FUGs proved inefficient when dealing with some lexical choice issues and we faced modularity problems when introducing additional knowledge sources such as a collocation dictionary [24].

To address these limitations, we propose to integrate explicit control annotations within FUGs. Specifically, we present two control tools that we have implemented in FUF: dependency directed backtracking and co-routine interface with external sources of constraints. In this paper, we first describe the control regime used for generation in FUG. We then introduce the `bk-class` and `external` tools and discuss how these new control features integrate within the general control flow of FUF.

2 STANDARD CONTROL IN FUG

FUG relies on the primitive operation of unification of functional descriptions (FDs) [13]. FDs are sets of pairs `(a v)`, called features, where `a` is an attribute and `v` is a value, either an atomic symbol or recursively an FD. An 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 value for the same attribute. 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 two specific constructs of FUG that are of importance for this paper: `alt` and `any`. `Alt` 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.

`Any` is a special value that 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 (cf [7] for a discussion). `Any` is therefore a meta-variable that triggers a delayed check.

During sentence generation, unification is used to add syntactic information from a functional unification grammar (FUG) to a semantic input, both represented as FDs. Figure 1 shows the input for the sentence *''Robinson scored 32 points''* and a FUG (grammar G1) expressing the mapping from semantic categories to syntactic categories. For example, G1 specifies that semantic actions are to be syntactically realized by clauses. When unifying the semantic input I1 with this FUG the following operations are performed: FUF picks branch 2 of the `alt` and merges it with the input. During the merging, the features underlined in Figure 1 get added to the result.

The semantic input is a structured representation. It consists of a toplevel predicate with embedded arguments. In the single unification shown in Figure 1 however, only the toplevel 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.

To understand how FUF proceeds at this point, we must define 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 Figure 1. Intuitively, constituents bring structure to functional descriptions.

To handle constituents, the complete unification procedure is:

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

Constituents therefore trigger recursion in FUGs.

Semantic input I1:

```
 ;; Predicate
((sem-cat action)
 (concept score)
 (tense past)

 ;; Arguments
 (agent ((sem-cat individual)
         (concept player)
         (name Robinson)))
 (medium ((sem-cat set)
          (concept point)
          (cardinal 32))))
```

Grammar G1:

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

Total FD after a single unification:

```
((sem-cat action)
 (concept score)
 (tense past)
 (synt-cat clause)
 (CSET (agent medium))
 (agent ((sem-cat individual)
         (concept player)
         (name Robinson)))
 (medium ((sem-cat set)
          (concept point)
          (cardinal 32))))
```

Figure 1: Example of unification

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: toplevel first, then `agent` and `medium`. The resulting FD at the end of the process is shown in Figure 2.

This control regime is driven by the structure of the semantic input FD, and not by an *a priori* syntactic structure described in the grammar. It is therefore similar in spirit to the semantic-head-driven algorithm presented in [21].

Total unification produces:

```
((sem-cat action)
 (concept score)
 (synt-cat clause)
 (CSET (agent medium))
 (agent ((sem-cat individual)
         (synt-cat proper-name)
         (concept player)
         (name Robinson)))
 (medium ((sem-cat set)
          (synt-cat np)
          (concept point)
          (cardinal 32)))
 (tense past))
```

Figure 2: Complete unification

3 BK-CLASS AND NON-STRUCTURAL CONSTRAINTS

The top-down regime implemented in FUF generally handles the semantic constraints found in the input efficiently. However, the input to a text generator must also include pragmatic constraints that are inherently non-structural. What makes these constraints non-structural is that they may be expressed at different levels of the syntactic tree. They can therefore trigger non-local backtracking, beyond the scope of a single constituent. Such non-local backtracking can be very inefficient. We illustrate this problem in the following example and introduce a control tool called `bk-class` to cope with it.

Consider a system reporting on the results of a basketball game and an input containing the three following constraints:

- **Semantic Predication**: Convey that the Denver Nuggets defeated the Boston Celtics by a 101-99 score.
- **Manner Qualification**: Convey that the victory was narrow.
- **Argumentative Orientation**: Convey (explicitly or implicitly) the low rating of the Denver Nuggets.

This configuration of constraints is correctly satisfied by the following sentences:

- *''The hapless Denver Nuggets edged the Boston Celtics 101-99.''*
- *''Against all odds, the Denver Nuggets nipped the Boston Celtics 101-99.''*

But the following sentences fail to satisfy the constraint combination for different reasons:

```
((sem-cat action)
 (alt (index on concept)
   ;; Map the concept game-result to a verb
   (((concept win)

     (alt (bk-class (AO MANNER))

          ;; The winner's rating is poor
       ((({AO} ((concept rating)
                (carrier {winner})
                (orientation -)
                (conveyed yes)))
          (lex ((alt ("stun" "surprise")))))

         ;; The score is narrow
         (({manner} ((concept narrow)
                     (conveyed yes)))
          (lex ((alt ("edge" "nip")))))

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

Figure 4: Fragment of the lexicon

1. *? The Denver Nuggets narrowly stunned the Boston Celtics 101-99.*
   [Ambiguous modification by *''narrowly''*]
2. *? Against all odds, the Denver Nuggets narrowly beat the Boston Celtics 101-99.*
   [Ambiguous interaction between adjuncts]
3. *The Denver Nuggets edged the Boston Celtics 101-99.*
   [The Argumentative Orientation is not conveyed]
4. *The Denver Nuggets stunned the Boston Celtics 101-99.*
   [The manner qualification is not conveyed]

The manner constraint can be realized by different means: (1) choice of a verb like *''edge''* or *''nip''*, (2) use of an adverb like *''narrowly''*. Similarly, the constraints on the argumentative orientation (AO) can be satisfied: (1) by choosing a verb like *''surprise''* or *''stun''*, (2) by including an adjective like *''hapless''*, (3) by adding an adverbial adjunct like *''surprisingly''* or *''against all odds''*. The argumentative constraint could also be expressed by conjoining the clause with a connective like *''but''* [8, 9].

In order to choose between these different options, we use the following heuristics:

- Use semantically rich lexical heads rather than modifiers in order to create concise sentences.
- Don't use multiple adverbial adjuncts to avoid unpredictable interaction.

The input FD shown in Figure 3 encodes the three constraints we want to satisfy. Figure 4 shows a lexicon specifying the mapping between concepts and lexical items. The fragment shows how different verbs impose constraints on the features `AO` or `manner`,[1] or no constraint for ''neutral'' verbs.

Assuming FUF's top-down regime, unification of the input of Figure 3 with the grammar of Figure 5 proceeds as follows: the toplevel information is mapped to a verb-group syntactic constituent. Semantic constituents are then mapped to syntactic functions: `winner` to subject, `loser` to object and `score` to an adjunct. These constituents are *structural* constraints.

In contrast, `AO` and `manner` are ''floating'' constraints: they can be realized at many different levels of the syntactic tree. They are *non-structural* constraints, so they are not mapped to syntactic constituents right away.

Figure 5 shows how the `manner` input constraint is handled by the grammar. The feature `{manner conveyed}` is set to `yes` when the manner constraint is realized by some linguistic device. In the first branch

[1] The notation ({AO} v) is an equation which means that the feature (AO v) is not embedded in the containing list but is a toplevel feature.

```
( ...

 (verb ...)

 (AO ...)

 (manner
  ((alt (bk-class manner)
       (
        ;; can be realized by other means - delay
        ((conveyed any))

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

  ... )
```

Figure 5: Handling of a non-structural constraint in the grammar

```
;; Semantic predication
((sem-cat action)
 (concept win)
 (tense past)
 (winner ((sem-cat individual)
          (concept team)
          (name Nuggets)))
 (loser ((sem-cat individual)
         (concept team)
         (name Celtics)))
 (score ((sem-cat quantity)
         (concept game-score)
         (winner-score 101)
         (loser-score 99))))

 ;; Manner constraint
 (manner ((sem-cat quality)
          (concept narrow)))

 ;; Argumentative constraint
 (AO ((sem-cat scale)
      (concept rating)
      (carrier {winner})
      (orientation -))))
```

Figure 3: Input FD expressing 3 constraints

of the `alt`, we first check whether the manner constraint has been handled without an adverb, *e.g.*, when the verb lexically carries manner. This check is implemented by the feature `(conveyed any)`. This first branch delays the decision to use an adverb with the `any` construct. This leaves a chance for the other devices to express the manner constraint. If however no other linguistic device can be found that satisfies the manner constraint, the grammar resorts to using an adverbial adjunct. The feature `(conveyed any)` therefore prevents the generation of semantically incomplete sentences like Examples 3 and 4 above.

The heuristic of avoiding multiple adverbs is implemented by including a single `adverb` constituent in the total FD[2]. The heuristic of preferring richer semantic heads over modifiers is implemented by the ordering of the branches in this `alt`: we first try without the adverb, and then only if necessary do we use one. The argumentative constraint is treated similarly.

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 `win` is lexicalized. We are now traversing the lexicon fragment in Figure 4 and first choose the verb ‘‘stun’’ which satisfies both the semantic predication and the argumentative constraint. We then map the semantic constituents to syntactic functions and proceed to the argumentative constraint. It is already satisfied by the verb, so no modifier needs to be introduced.

Finally comes the manner constraint. We first delay the use of an adverb with the `any` construct. The unifier completes the traversal of the constituents top-down. It eventually checks the `any` construct and finds that the manner constraint has not been accounted for. 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 we blindly backtrack, search is maximized. Since we do not know in advance where in the syntactic structure the floating constraint of manner can be satisfied, we need to delay

[2]Recall that an FD can contain only a single occurrence of a given feature and that each feature has only a single value.

the decision whether to use an adverb until the end of the traversal. So there is no way to detect failure before this point.

To avoid the cost of a blind backtracking, we introduce the `bk-class` construct. `bk-class` implements a version of dependency-directed backtracking [5] specialized to the case of FUF. `bk-class` 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 imposes the value of the feature `{manner conveyed}` to be instantiated and the actual feature is not. The path `{manner conveyed}` defines the *address of the failure*.

The idea is that the location of certain failures can be used to identify the only decision points in the backtracking stack that could have caused the failure. This identification requires 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: `(def-bk-class manner {manner conveyed})` specifies that the path {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 Figures 4 and 5.

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 (Figure 5). If this last option fails again, we backtrack up to the choice of verb in the lexicon (Figure 4). 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 to independently express each constraint in the FUG.

In the case of non-structural constraints like argumentation or manner, the dependency-directed mechanism implemented in FUF with `bk-class` nicely complements a general top-down control regime. This mechanism improves FUGs efficiency while preserving their desirable properties - declarativity and bidirectional constraint satisfaction.

4 EXTERNAL AND MODULARITY

Surface realization consists in 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 [15] and [20]. 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 [11, p.73].
- *Portability*: the grammar rules, the grammatical dictionary [3] and to some extent the collocation dictionary [22] 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 [4], this is also the case for the knowledge base 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 be non-modular and thus to hinder 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 meets a feature of the form `(a #(external F))` it performs the following operation: unification is suspended; the external function F (a LISP function returning an FD) is evaluated; the return value becomes the new value of the attribute; unification resumes using this new value. For example, if `F` returns the value `b`, the next feature to be unified is `(a b)`. `External` therefore allows the dynamic expansion of a FUG at unification-time.

`External` enhances FUF in multiple ways:

- It provides a co-routine control structure to interact with external processes.
- It enforces an information-hiding principle between different knowledge source.
- 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 [1] implemented a mechanism where a FUG and a content planner cooperated to generate referring expressions. `External` is a generalization of this mechanism. With PAULINE [12], Hovy advocated interleaving surface realization with content planning pervasively. EXTERNAL can implement such an interleaving in FUF, while benefiting from the declarative nature of FUGs. Finally, PENMAN [14], while traversing a systemic linguistic network, accesses its environment by calling functions called *inquiries*. `External` provides a similar facility in the context of FUGs.

5 CONCLUSION AND FUTURE WORK

When designing control tools, we want to address the following general issues:

- Modularity: we want to maintain a clean separation between different knowledge sources.
- Efficiency: we want the control mechanism to take advantage of available knowledge to reduce search.

We presented `external` and `bk-class`, two control tools augmenting the general semantic top-down regime of FUGs to address these control issues in text generation. These control tools have been implemented in FUF and tested on several applications (COMET [2], a system generating explanations in a multi-media setting, COOK [22] a sentence-generation system in the domain of stock market dealing with collocations and ADVISOR [8], an explanation system dealing with argumentative constraints).

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

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 non-structural constraint. Naish in [18, p.59] lists heuristics to improve efficiency in search, including ''detect failure early'' and ''avoid failure.'' We have shown in Section 3 that there is good linguistic motivation that makes an early detection of failure for ''floating'' non-structural constraints very difficult. In such cases, `bk-class` implements the heuristic of avoiding failure.

When the interaction of several non-structural constraints is further investigated (*e.g.*, including collocations), a more sophisticated mechanism for delaying and awakening choices is desirable. We are currently implementing such a tool for freezing decisions called `alt*`.

References

[1] Appelt, D.E.
Planning English Sentences.
Cambridge University Press, Cambridge, UK, 1985.

[2] McKeown, K., Elhadad, M., Fukumoto, Y., Lim, J., Lombardi, C., Robin, J. and Smadja, F.
Language Generation in COMET.
In Mellish, C. and Dale, R. and Zock, M. (editor), *Current Research in Language Generation*, pages 103-140. Academic Press, London, UK, 1990.

[3] Cumming, S. and Albano, R.
A guide to lexical acquisition in the Janus system.
Technical Report ISI/RR-85-162, ISI, Marina del Rey, CA, 1986.

[4] Dale, R.
Generating referring expressions in a domain of objects and processes.
PhD thesis, University of Edinburgh, 1988.

[5] de Kleer, J., Doyle, J., Steele, G.L.Jr, Sussman, G.J.
Explicit Control of Reasoning.
Artificial Intelligence: an MIT Perspective.
MIT Press, 1979, pages 93-116.

[6] Elhadad, M.
Types in Functional Unification Grammars.
In *Proceedings of 28th Meeting of the ACL (ACL'90)*. Pittsburgh, 1990.

[7] Elhadad, M.
The FUF Functional Unifier: User's manual (Version 3.0).
Technical Report CUCS-408-88, Columbia University, June, 1990.

[8] Michael Elhadad.
Constraint-based Text Generation: Using local Constraints and Argumentation to Generate a Turn in Conversation.
Technical Report CUCS-003-90, Columbia University, 1990.

[9] Elhadad, M. and K.R. McKeown.
Generating Connectives.
In *Proceedings of COLING'90 (Volume 3)*, pages 97-101. Helsinki, Finland, 1990.

[10] Elhadad, M., Seligmann, D.D., Feiner, S. and McKeown, K.R.
A Common Intention Description Language for Interactive Multi-media Systems.
In *Presented at the Workshop on Intelligent Interfaces, IJCAI 89*. Detroit, MI, 1989.

[11] Halliday, M.A.K.
System and Function in Language.
Oxford University Press, London, 1976.

[12] Eduard Hovy.
Generating natural language under pragmatic constraints.
PhD thesis, Yale University, 1987.

[13] Kay, M.
Functional Grammar.
In *Proceedings of the 5th meeting of the Berkeley Linguistics Society*. Berkeley Linguistics Society, 1979.

[14] Mann, W.C.
An overview of the Nigel Text Generation Grammar.
Technical Report ISI/RR-83-113, USC/ISI, April, 1983.

[15] Matthiessen, C.M.
The organization of the environment of a text-generation grammar.
Natural Language Generation: New Results in Artificial Intellligence, Psychology and Linguistics.
Martinus Ninjhoff Publishers, 1987.

[16] McKeown, K.R.
Text Generation: Using Discourse Strategies and Focus Constraints to Generate Natural Language Text.
Cambridge University Press, Cambridge, England, 1985.

[17] McKeown, K. and M. Elhadad.
A Contrastive Evaluation of Functional Unification Grammar for Surface Language Generators: A Case Study in Choice of Connectives.
Natural Language Generation in Artificial Intelligence and Computational Linguistics.
Kluwer Academic Publishers, 1991.

[18] Naish, Lee.
Lectures Notes in Computer Science. Volume 238: *Negation and Control in Prolog.*
Springer Verlag, 1985.

[19] Paris, C.L.
The Use of Explicit User models in Text Generation: Tailoring to a User's level of expertise.
PhD thesis, Columbia University, 1987.
To appear Frances Pinter publ. in the series *Communications in Artifical Intelligence*, Steiner and Fawcett (Eds).

[20] Robin, J.
Lexical Choice in Natural Language Generation.
Technical Report CUCS-040-90, Columbia University, 1990.

[21] Shieber, S.M., van Noord, G., Moore, R.M. and Pereira F.C.P.
A Semantic Head-Driven Generation Algorithm for Unification-based Formalisms.
In *Proceedings of the 27th ACL*, pages 7-17. ACL, Vancouver, British Columbia, Canada, 1989.

[22] Smadja, F.
Microcoding the Lexicon with Co-Occurrence Knowledge.
In Uri Zernik (editor), *Lexical Acquisition: Using on-line resources to build a lexicon*. Lawrence Erlbaum, 1991.
In press.

[23] Smadja, F. and McKeown, K.
Automatically Extracting and Representing Collocations for Language Generation.
In *Proceedings of the 28th annual meeting of the ACL*. Pittsburgh, PA, June, 1990.

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