Interfacing Constraint-Based Grammars and Generation Algorithms

Stephan Busemann
DFKI GmbH
Stuhlsatzenhausweg 3, D-66123 Saarbrücken, Germany
e-mail: `busemann@dfki.de`

Abstract

Constraint-based grammars can, in principle, serve as the major linguistic knowledge source for both parsing and generation. Surface generation starts from input semantics representations that may vary across grammars. For many declarative grammars, the concept of derivation implicitly built in is that of parsing. They may thus not be interpretable by a generation algorithm. We show that linguistically plausible semantic analyses can cause severe problems for semantic-head-driven approaches for generation (SHDG). We use SEREAL, a variant of SHDG and the DISCO grammar of German as our source of examples. We propose a new, general approach that explicitly accounts for the interface between the grammar and the generation algorithm by adding a control-oriented layer to the linguistic knowledge base that reorganizes the semantics in a way suitable for generation.

1 Introduction

The relation between declaratively represented grammars and control structures that can process them is often described along the following lines: The declarative representation of grammars allows a grammar writer to describe the well-formed sentences of a language in an non-directional way. Such grammars can be used for both parsing and generation and are called reversible. On the other hand, algorithms for processing grammars are free from language-specific stipulations and can operate on different grammars within the same formalism.

Unfortunately, this picture is too superficial and needs some discussion and refinement. The declarative representation of grammars is a necessary prerequisite for ensuring reversibility, but it is far from sufficient.

This paper reports on a case study, in which a declaratively represented grammar developed by linguists and used for parsing was employed for generation. The generation algorithm used is a variant of Semantic-Head-Driven Generation (SHDG) (Shieber et al., 1990). It interprets a large constraint-based grammar of German developed for the DISCO system (Dialogue System for Cooperating agents) (Uszkoreit et al., 1994). SHDG is one of the most widespread algorithms for sentence realization with constraint-based grammars. It is largely theory-independent and has been used for Head-Driven Phrase Structure Grammars (HPSG), Definite Clause Grammars, and Categorial Unification Grammars. Since its publication, SHDG had to compete with other algorithms (e.g. (Russell et al., 1990), (Strzalkowski, 1994), (Martinovic and Strzalkowski, 1992)) which led to numerous ways of improving the basic procedure.

A major question remained unsolved (and it is unsolved for other algorithms as well), namely that of the algorithm's requirements on the properties of the grammar used. In previous work, Shieber imposed a condition on "semantic monotonicity" that holds for a grammar if for every phrase the semantic structure of each immediate subphrase subsumes some portion of the semantic structure of the entire phrase (Shieber, 1988, p. 617). Semantic monotonicity is very strict and could be relaxed in SHDG: It was shown that semantically non-monotonic grammars can be processed by SHDG. It is a yet open question whether all semantically monotonic grammars can be processed by SHDG and what the class of SHDG-processable grammars is.

Using the linguistically well motivated semantics of DISCO as a sample input language to SHDG, we show that there are semantically monotonic grammars that cannot be processed directly by SHDG. The difficulties encountered are of a general kind, and a general approach for solving them is presented that explicitly accounts for the interface between the grammar and the generation algorithm by adding a modular, control-oriented layer to the linguistic knowledge base that represents a reorganization of the semantics in a way suitable for generation. Moreover, we present the specific solution for the grammar in hand.

The kind of problem investigated in this paper relates to the fundamental question of how to organize a modular system consisting of linguistic knowledge (a grammar) and control knowledge (parser or generator). It turns out that declarative grammars contain hidden assumptions about processing issues that need to be made explicit.

2 SHDG and the Grammar Interface

Without loosing generality we assume that the grammar has a semantics layer and that a generator input expression is an element of the semantic representation language encoded by the grammar. The generator is guided by its input. Thus we refer to the semantics layer as the *essential feature.*

We now briefly review some essential points of SHDG (Shieber et al., 1990). The algorithm is centered around the notion of a *pivot* node, which provides an essential feature specification from which it first generates all descendants in a top-down manner, and then tries to connect the newly generated subtree to a higher node (or the root node) in bottom-up fashion. Both generating descendants and connecting to higher nodes involves the application of grammar rules. Correspondingly, rules are subdivided into two classes: *chain rules* are used for bottom-up connection while *non-chain rules* are applied for top-down expansion. Chain rules differ from non-chain rules in that their left-hand side essential feature is identical to the essential feature of one of their right-hand side elements. This element is called the "semantic head" of the chain rule. Lexical entries are non-chain rules in a trivial way since they have no categorial right-hand side elements.

The only specific assumption SHDG makes about a grammar is that chain rules and their semantic heads can be identified. However, the property of being a chain rule (or non-chain rule) is often assigned by the grammar writer on purely linguistic grounds although it determines the processing strategy: If the set of chain rules happens to be empty, SHDG operates strictly top-down. If the set of non-chain rules consists of lexicon entries only, SHDG behaves like a bottom-up generator. Having the linguist unconsciously influence the processing strategy of SHDG can lead to uninterpretable grammars, as we will show below.

We now introduce some basic assumptions about grammars. A grammar induces a context-free backbone and has separate layers to represent morphological, syntactic, and semantic properties of categories. We assume furthermore that the generator can be told how to identify mother and daughter categories of grammar rules.

It has always been a matter of discussion how a surface generator should cope with the presence or absence of essential feature specifications. Since a closer investigation of this issue is beyond the scope of this paper, we assume that the input structure and the semantics of the sentence to be generated are equal, i.e. they subsume each other.

A generator must *terminate* with results on all allowable input. The under-specification of the essential feature at execution time is a well-known phenomenon (Russell et al., 1990). It can show up during top-down expansion of a grammar rule that does not share the essential features of the daughters with parts of the mother. Non-termination or failure to find a derivation will result. To avoid underspecification, the following condition on generator/grammar pairs ensures successful recursive applicability of the generation procedure:

> **Essential Feature Specification Condition (EFSC):** The essential feature must specify exactly the constituent to be generated at the time the generation procedure is executed on it.

This requirement captures what should be stated independently of particular generators and grammars. For individual generator/grammar pairs, EFSC needs to be concretized. Any such concretization must take the specific algorithm into account. For instance, specifications of EFSC involving SHDG depend on the order in which nodes of a local tree are recursively expanded. (Shieber et al., 1990) quite arbitrarily assume a strict left-to-right processing of non-semantic-head daughter nodes. EFSC is easily violated by a daughter of a non-chain rule that influences the essential feature of a preceding daughter.

For the purpose of the present paper, we make a simplification by assuming that "exact specification" implies that the essential feature specification is subsumed by (i.e. is as specific as) the corresponding part of the input structure. While this is a sufficient condition on successful termination, the necessary one may indeed be weaker. It depends on the architecture of the grammar, from which we want to abstract away.

3 The System Setup

This section introduces the generator/grammar pair used for the present study. After a sketch of our variant of SHDG we discuss the semantics layer of the constraint-based grammar of German to the extent necessary to demonstrate violation of EFSC and to describe a solution.

3.1 The SeReal system

The SeReal (Sentence Realizer) is a Common Lisp SHDG implementation that uses kernel components of the Disco NL understanding system (Uszkoreit et al., 1994).

Disco is a linguistic core engine capable of analyzing NL sentences as quasi-logical form representations that can subsequently be submitted to further semantic analysis. The Disco grammar is encoded in TDL (Krieger and Schäfer, 1994), a powerful type definition language and type inference mechanism

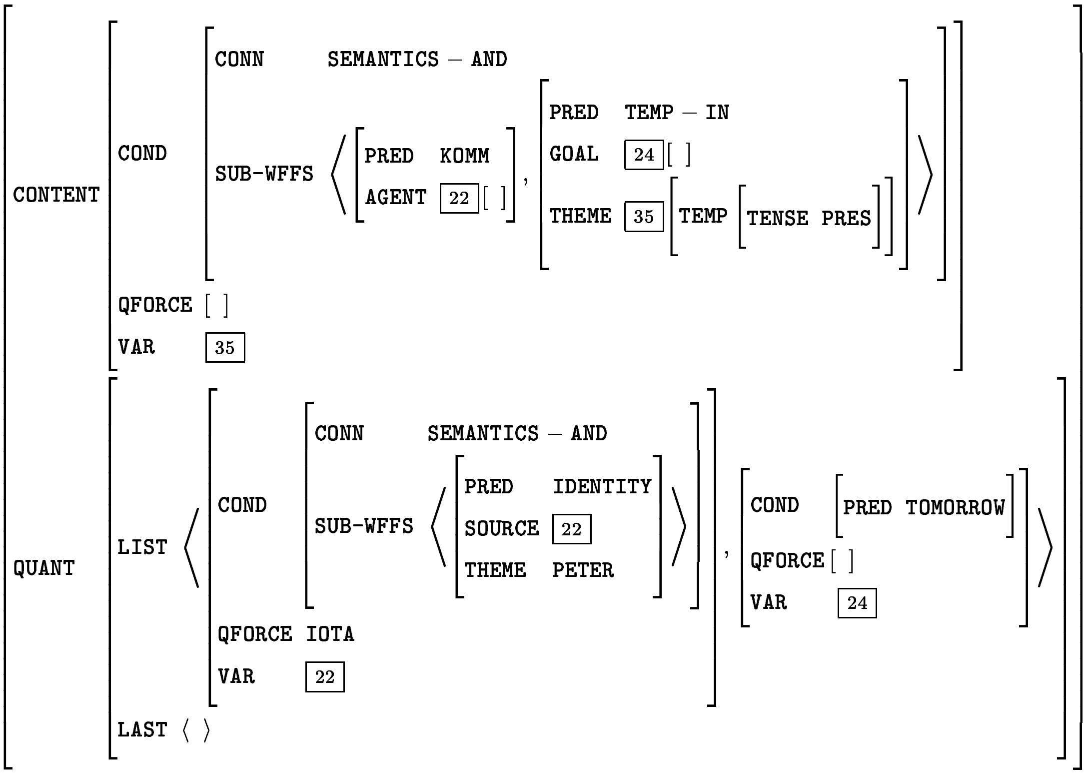

Figure 1: Semantic Feature Structure for *Peter kommt morgen* [Peter arrives tomorrow].

for feature structures. The basic processing engine is the feature constraint solver UDiNe, which is used to perform (destructive) unification during parsing and generation. A mapping between word forms and morpho-syntactically annotated word stems is achieved by the Morphix-3 system (Finkler and Neumann, 1988). SeReal is integrated into the Disco system to the extent that it uses the same grammar, UDiNe, TDL, and Morphix-3. It can be fed with the parser's semantics output and thus serve as a useful grammar development tool.

A special mechanism had to be developed for efficient lexicon access. The SHDG algorithm simply assumes all lexicon entries to be available as non-chain rules. This is, however, not advisable for large lexicons. Rather, only the relevant entries should be accessed. Therefore, SeReal indexes the lexicon according to semantic information. Consider, for instance, the semantic representation in Figure 1.[1] Lexical indices usually are semantic predicates denoted by the `PRED` feature, e.g. `KOMM` is the index for the main verb (*arrive*). Exceptions include determiners, which are indexed according to the value of `QFORCE` and proper names, which are indexed according to the value of `THEME`. A priority system on indices (`THEME` > `QFORCE` > `PRED`) reduces the number of accessible indices. This way an index points to very few lexicon entries.[2] Indices are retrieved as values of some path in the essential feature specification. Insertion of an entry into a derivation requires its essential feature to subsume the input structure to avoid introducing spurious semantic specifications.

Clearly both indices and path descriptions are grammar dependent and form a part of the interface between SeReal and the Disco grammar. In Figure 1, the following indices are used to access lexicon entries: `KOMM, PETER, TEMP-IN`.

The algorithm has been criticized for not terminating on left-recursive rules (Strzalkowski, 1994). Under the assumption of semantic monotonicity, the determination of a pivot can be conditioned by a

---

[1] This is a simplified version of a semantic representation taken from a parse with the Disco grammar. For presentation purposes we adopt the familiar matrix notation for feature structures. < and > are print macros for lists that expand into the common feature structure notation for lists (cf. (Shieber, 1986, page 29)). Although TDL defines *typed* feature structures, we omit type information here as it is not relevant.

[2] This depends on how many lexemes carry the same index. Usually we have one to three, in rare cases up to fifteen, entries per index.

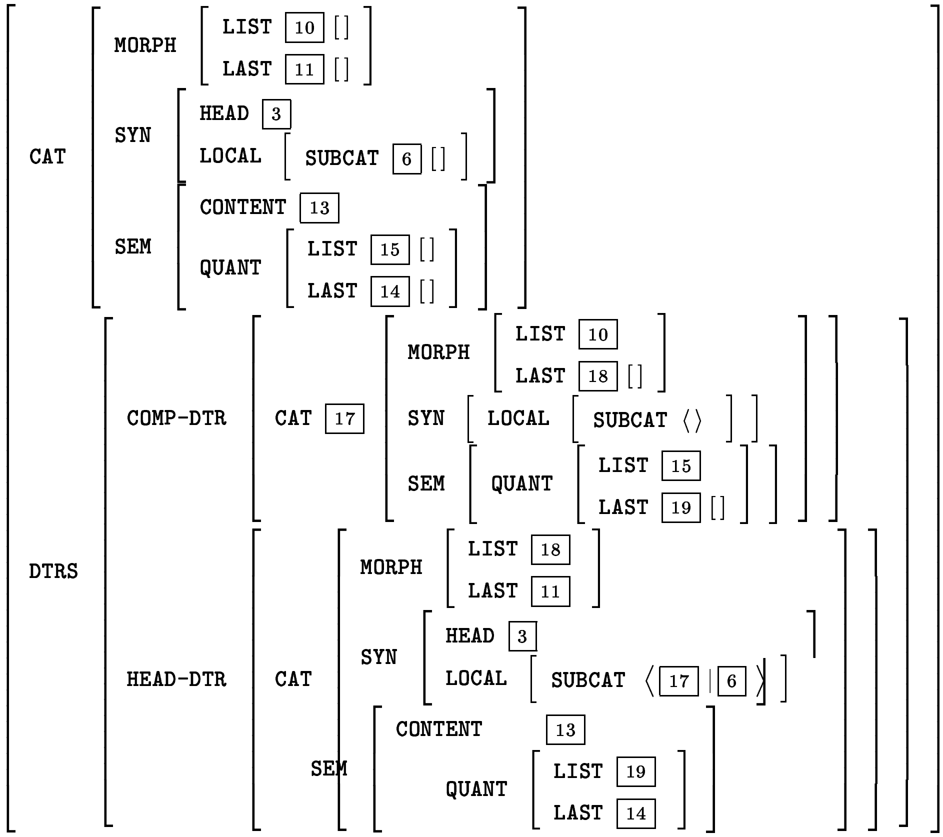

Figure 2: A Head-Complement Rule (simplified for expository purposes).

check for semantic content. If the semantics is "empty" (i.e., it corresponds to the top feature structure), processing fails and alternative possibilities have to be explored. Since left recursion occurs only in top-down direction, we are dealing with non-chain rules, which ensures that the semantics of a right-hand side element differs from that of the left-hand side. Semantic monotonicity ensures that it is "smaller" in some sense, thus guaranteeing termination.

(Martinovic and Strzalkowski, 1992) criticized the possible failure of top-down expansion due to the strict left-to-right processing of the list of right-hand side elements. Since the instantiation of the semantics of some right-hand elements can depend on the previous successful expansion of others, a strict order that does not consider such relations is inadequate. In SeReal, the left-most right-hand side element of a rule is expanded first that has a non-empty semantics instantiated.

3.2 The Disco semantics layer

The Disco grammar is a semantically monotonic lexicalized, HPSG-style grammar of German with about 20 rules, 13 of them binary. The remaining ones are unary (lexical) rules that serve to introduce syntactic features for lexemes in particular environments. For instance, verb lexemes can be made finite or infinite, adjectives can be made attributive or predicative. The binary rules account for complement and adjunct realization.

The development of the Disco grammar was, as many others, based on purely linguistic motivations. Although a declarative representation is used, the concept of derivation implicitly built in is that of (bottom-up) parsing. Again, this is common. The parsing view of the grammar developer influences the goals that a semantic representation should fulfill. The Disco semantics layer should

- represent a linguistically well motivated (surface) propositional semantics of NL sentences,
- provide the interface to subsequent non-compositional, extra-grammatical semantic interpretation (e.g. anaphora resolution, scope disambiguation), and
- represent the essential feature for grammar-based sentence realization.

The semantics layer corresponds to quasi-logical forms (Alshawi, 1992) that are defined through the grammar and represented with help of feature structures (Nerbonne, 1992). The relevance of the surface ordering of complements and adjuncts during later semantic processing made it necessary to encode ordering information at the semantics layer. This is reflected by the `QUANT` feature, which contains a list of the semantics of the complements and adjuncts in the order they occur at the surface. The relations between them are expressed by the `CONTENT` feature with help of the `VAR` feature.

Consider as an example the semantics structure in Figure 1. `QUANT` has two elements, the first one representing the proper name and the second one the temporal adverb *tomorrow*. `CONTENT` represents a `COND`ition on the meaning consisting of a conjunction of sub-formulae. The first formula represents a one-place predicate `KOMM`, the argument of which points, via `VAR`, into the first element of the `QUANT` list. The second sub-formula represents a two-place predicate `TEMP-IN`. Its first argument points into the second element of `QUANT`, and its second argument relates to the whole `CONTENT` feature. Thus the predicate is to be interpreted as a temporal sentential modifier.

Semantic information mainly originates from lexical entries. A few general principles of feature distribution are represented with the grammar rules. Figure 2 shows a head-complement rule with the complement being the first element of the head's subcategorization list. The complement is preceding the head (not shown). `CONTENT` is shared between the mother (`CAT`) and the head daughter. In a rule's left-hand side constituent, `QUANT` denotes the concatenation of the `QUANT` values of the sequence of right-hand side elements.

List concatenation is encoded using difference lists. Thus it is not necessary to use functional feature values such as `append`. The difference list type built into in TDL denotes a list L by defining a list L1 under the feature `LIST` and another list L2 under the feature `LAST` such that L2 is a tail of L1 and the concatenation of L and L2 yields L1. This can easily be achieved by choosing appropriate coreferences.

In the case of bottom-up processing, this mechanism is used like a stack: at the mother node, the `QUANT` feature of the complement semantics has been pushed onto the list of elements collected so far (at the head daughter).

4 A Violation of EFSC

Investigation of the grammar rules shows that there are no binary chain rules since the `QUANT` feature within `SEM` differs at all nodes of a rule (cf. Figure 2). With the resulting top-down strategy the `QUANT` list at the mother node must be split into two sublists in order to instantiate the `QUANT` lists of the daughter nodes. This is a nondeterministic problem that, given the present implementation of difference lists, leads to under-specification.

Unification of some input semantics with the mother node (in Figure 2 under `CAT.SEM`[3]) does not specify how the `QUANT` list should be split, i.e. the `QUANT.LAST` feature of the `COMP-DTR` semantics, which is shared with the `QUANT.LIST` feature of the `HEAD-DTR` semantics, is not affected at all by this unification operation. Any further expansion steps using similar rules will not specify the semantics any further, and hence non-termination results.[4]

This problem is not specific to the Disco grammar. Difference lists are a common descriptive device used in many constraint-based grammars. For instance, the same problem arises with the Minimal Recursion Semantics, a framework for semantics within HPSG, which was developed to simplify transfer and generation for machine translation (Copestake et al., 1995; Copestake et al., 1997).

Neither is the problem specific to SeReal or SHDG. It is due to the fact that there is no inverse function for list concatenation, causing the simulating difference list mechanism to fail on splitting lists.

5 Reorganizing Semantic Information

Whenever a grammar/generator pair violates EFSC, two basic repair strategies offer themselves as remedies: Either the generator is modified to account for the grammatical analysis, or the grammar is adapted to the needs of the generator.

$$\left[\begin{array}{ll} \texttt{SEM} & \left[\texttt{CONTENT}\ \boxed{1}\ [\,]\right] \\ \texttt{GEN} & \left[\begin{array}{ll} \texttt{TODO} & \left[\begin{array}{ll}\texttt{LIST} & \boxed{2}\ [\,] \\ \texttt{LAST} & \boxed{3}\ [\,]\end{array}\right] \\ \texttt{PIVOT} & \left[\begin{array}{ll}\texttt{CONTENT-GEN} & \boxed{1} \\ \texttt{QUINPUT} & \left[\begin{array}{ll}\texttt{LIST} & \boxed{2} \\ \texttt{LAST} & \boxed{4}\ [\,]\end{array}\right]\end{array}\right] \\ \texttt{DONE} & \left[\begin{array}{ll}\texttt{LIST} & \boxed{3} \\ \texttt{LAST} & \boxed{4}\end{array}\right]\end{array}\right]\end{array}\right]$$

Figure 3: The Organization of the `GEN` Layer.

Grammar writing should be guided by linguistic adequacy considerations rather than by algorithmic issues. Linguistically plausible analyses should not

---

[3] We use the period between feature names to denote feature path descriptions.

[4] It may be argued that the `CONTENT` feature could serve as a pivot. It is indeed shared between mother and head in most rules, which would then be chain rule candidates. However, semantic information necessary to guide the generation of many phrasal constituents may be represented only by `QUANT`.

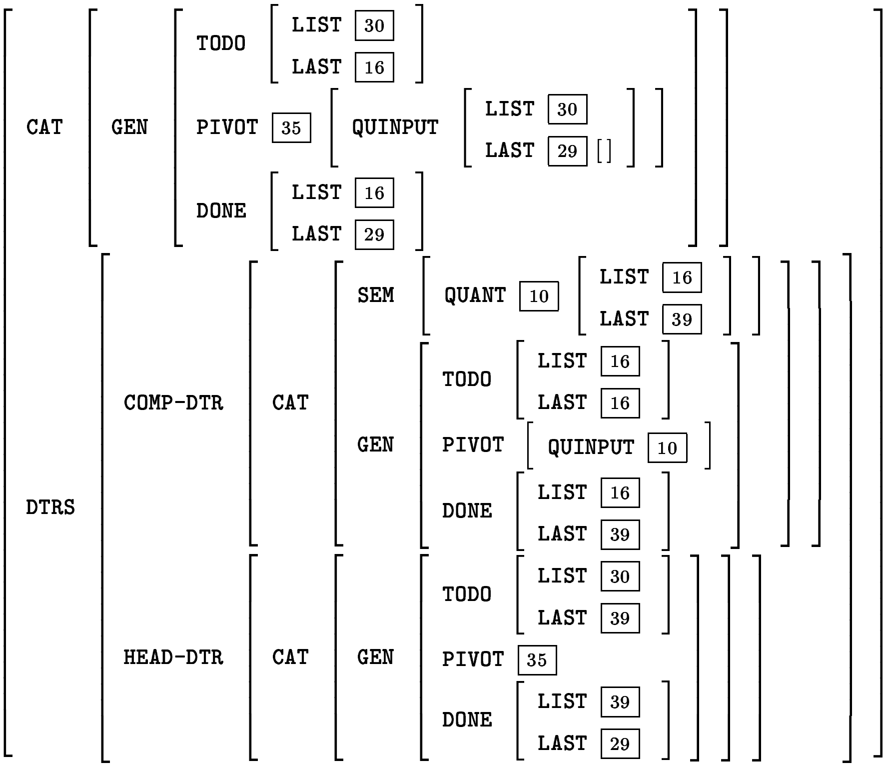

Figure 4: The GEN Feature in a Head-Complement Rule.

be rejected because they are not processed by the generator used. On the other hand, designers of generation (or parsing) algorithms want to create generic tools that can be used for large classes of grammars. Such algorithms, including those of the SHDG type, should not be geared towards a particular grammar. Moreover, in a large grammar, processing problems may occur with several phenomena, and solving them either way might eventually sacrifice the modularity of the grammar and the generator.

In conclusion, neither of the two ways is satisfactory. A third strategy is to design generators in such a way that they comply with a particular grammar theory. (Wilcock and Matsumoto, 1998) describe modifications of bottom-up generation (van Noord, 1990) to comply with the current version of HPSG (Pollard and Sag, 1994). Given the many variants of linguistic theories used in implemented systems world-wide, this strategy will probably in most cases boil down to adapting generators to grammars that adhere to a "local" version of a theory.

In this contribution we present a novel approach that complements a single grammar by an explicit and modular interface layer that restructures the semantic information in such a way that it supports bottom-up processing within SEREAL. This method improves over previous approaches in various ways:

- The interface is defined declaratively;
- Reversibility properties of the grammar are preserved;
- The modularity of the grammar and the generator are preserved.

This layer, GEN, is assigned to every category of the grammar (cf. Figure 3). We make use of the constraint-based formalism (here: TDL) in defining GEN and relating it to the grammar. It is very important to notice that its definition does not modify the grammar, rather a new module is added to it. Since semantic information is not constrained, but just restructured in GEN, reversibility properties of the grammar are not touched. Parsing results are completely independent from the presence of GEN. Since the restructuring is achieved by using coreferences with the parts of the semantic layer, generation uses the same kind of semantic information as parsing. Hence, SEREAL will deliver all sentences for a semantic representation restructured in GEN that yield that semantic representation when they are parsed.

In the case at hand, we relate GEN to the DISCO semantics as follows. Within GEN we define a new essential feature, `PIVOT`, that shares the semantic content (under `CONTENT-GEN`) and contains the `QUANT` list of the input (under `QUINPUT`). We specify explicitly the sublist of `QUINPUT` covered by the subtree represented by the category at hand using the list `DONE`, and we also note the list of remaining elements that still need to be processed (`TODO`). This is encoded using difference lists.

The binary grammar rules are extended as follows (Figure 4 shows the `GEN` feature added to the rule in Figure 2). Mother and head daughter share their `PIVOT` features, which yields us chain rules (and the desired bottom-up processing strategy). Obviously the mother's `DONE` list must be the concatenation of all daughters' `DONE` lists. Moreover, the complement daughter's `TODO` list must be empty, which is why `QUINPUT` and `DONE` coincide. `QUINPUT` of the complement daughter is shared with `SEM.QUANT`. It is completely specified after the subtree represented by the head daughter has been completed.

6 Conclusion

Interfaces between constraint-based grammars and generation systems ultimaltely are defined in a very specialized way. In view of the disadvantages of current approaches dealing with EFSC violations, we have introduced into the descriptive framework a new, control-oriented layer of representation, `GEN`, that reorganizes semantic information in such a way that it does not violate EFSC for the generation algorithm used.

`GEN` is the essential feature of a generation procedure and serves to define the interface between a grammar and a generator. This way, the interface is explicitly and declaratively defined. Grammars developed independently of a specific generator can be adapted quickly without changing them. Different interfaces can adapt a grammar to different generators.

Besides its architectural advantages, this approach has considerable practical benefits compared to compilation methods. It uses the same representational means that serve for the implementation of the grammar. If a grammar writer chooses to modify the encoding of certain linguistic phenomena, potential clashes with the interface definitions can be detected and removed more easily.

The method is generally applicable in constraint-based frameworks. The `GEN` layer must be defined explicitly for every generator/grammar pair. Depending on whether and where EFSC is violated, `GEN` may just co-specify the semantics (the trivial case), or reconstruct the semantics in an EFSC-compatible fashion. An instance of the latter was described above for the DISCO grammar and SEREAL. If a different generator is chosen for the DISCO grammar, neither the algorithm nor the grammar needs to be modified. The same holds true, if SEREAL was to interpret a different grammar. In both cases, it is the definition of `GEN` that would have to be replaced.

The techniques presented are implemented in TDL and CommonLisp within the SEREAL system.

Acknowledgments

This work has been partially supported by a grant from The German Federal Ministry for Research and Technology (FKZ ITW 9402). I am grateful to Feiyu Xu for implementing a first version of the interface, and to Jan Alexandersson, Edmund Grimley Evans, and Harald Lochert for implementing parts of the SEREAL system.

References

Hiyan Alshawi, editor. 1992. *The Core Language Engine.* ACL-MIT Press Series in Natural Language Processing. MIT Press, Cambridge MA.

Ann Copestake, Dan Flickinger, Robert Malouf, Susanne Riehemann, and Ivan A. Sag. 1995. Translation with Minimal Recursion Semantics. In *Proc. 6th International Conference on Theoretical and Methodological Issues in Machine Translation*, Leuven.

Ann Copestake, Dan Flickinger, and Ivan A. Sag. 1997. Minimal recursion semantics: An introduction. Available at `ftp://csli-ftp.stanford.edu/linguistics/sag/mrs.ps.gz`.

Wolfgang Finkler and Günter Neumann. 1988. Morphix: A fast realization of a classification–based approach to morphology. In H. Trost, editor, *Proceedings der 4. Österreichischen Artificial–Intelligence Tagung, Wiener Workshop Wissensbasierte Sprachverarbeitung*, pages 11–19, Berlin, August. Springer.

Hans-Ulrich Krieger and Ulrich Schäfer. 1994. *TDL*—a type description language for constraint-based grammars. In *Proceedings of the 15th International Conference on Computational Linguistics, COLING-94*, Kyoto, Japan.

Miroslav Martinovic and Tomek Strzalkowski. 1992. Comparing two grammar-based generation-algorithms: A case study. In *Proceedings of the 30th Annual Meeting of the Association for Computational Linguistics*, pages 81–88, Newark, Delaware.

John Nerbonne. 1992. Constraint-based semantics. In Paul Dekker and Martin Stokhof, editors, *Proceedings of the 8th Amsterdam Colloquium*, pages 425–444. Institute for Logic, Language and Computation. Also available as Research Report RR-92-18, Deutsches Forschungszentrum für Künstliche Intelligenz, Saarbrücken, Germany.

Carl J. Pollard and Ivan A. Sag. 1994. *Head Driven Phrase Structure Grammar.* Studies in Contemproary Linguistics. CSLI, Stanford & University of Chicago Press.

Graham Russell, Susan Warwick, and John Carroll. 1990. Asymmetry in parsing and generating with unification grammars: Case studies from ELU. In *Proc. Conf. of the 28th Annual Meeting of the ACL*, pages 205–211., Pittsburgh.

Stuart M. Shieber, Gertjan van Noord, Robert C. Moore, and Fernando C. N. Pereira. 1990. A semantic-head-driven generation algorithm for unification-based formalism. *Computational Linguistics*, 16(1):30–42.

Stuart M. Shieber. 1986. *An Introduction to Unification-Based Approaches to Grammar*, volume 4 of *CSLI Lecture Notes.* Stanford University, Stanford (CA).

Stuart M. Shieber. 1988. A uniform architecture for parsing and generation. In *Proceedings of the 12th International Conference on Computational Linguistics and the 24th Annual Meeting of the Association for Computational Linguistics*, pages 614–619, Budapest, Hungary, August 22-27.

Tomek Strzalkowski. 1994. A general computational method for grammar inversion. In Tomek Strzalkowski, editor, *Reversible Grammars in Natural Language Processing*, pages 175–200. Kluwer, Boston, Dordrecht, London.

Hans Uszkoreit, Rolf Backofen, Stephan Busemann, Abdel Kader Diagne, Elizabeth A. Hinkelman, Walter Kasper, Bernd Kiefer, Hans-Ulrich Krieger, Klaus Netter, Günter Neumann, Stephan Oepen, and Stephen P. Spackman. 1994. DISCO–An HPSG-based NLP System and its Application for Appointment Scheduling. In *Proceedings of the 15th International Conference on Computational Linguistics, COLING-94*, Kyoto, Japan.

Gertjan van Noord. 1990. An overview of head-driven bottom-up generation. In Robert Dale, Chris Mellish, and Michael Zock, editors, *Current Research in Natural Language Generation.* Academic Press, London.

Graham Wilcock and Yuji Matsumoto. 1998. Head-driven generation with HPSG. In *36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics. Proceedings*, pages 1393–1396, Montreal, Canada.