Best-First Surface Realization

Stephan Busemann

RR-96-05
Research Repo

Deutsches
Forschungszentrum
für Künstliche
Intelligenz GmbH

**Research
Report**

RR-96-05

Best-First Surface Realization

Stephan Busemann

May 1996

**Deutsches Forschungszentrum für Künstliche Intelligenz
GmbH**

Postfach 20 80
67608 Kaiserslautern, FRG
Tel.: + 49 (631) 205-3211
Fax: + 49 (631) 205-3210

Stuhlsatzenhausweg 3
66123 Saarbrücken, FRG
Tel.: + 49 (681) 302-5252
Fax: + 49 (681) 302-5341

Deutsches Forschungszentrum
für
Künstliche Intelligenz

The German Research Center for Artificial Intelligence (Deutsches Forschungszentrum für Künstliche Intelligenz, DFKI) with sites in Kaiserslautern and Saarbrücken is a non-profit organization which was founded in 1988. The shareholder companies are Atlas Elektronik, Daimler-Benz, Fraunhofer Gesellschaft, GMD, IBM, Insiders, Mannesmann-Kienzle, Sema Group, Siemens and Siemens-Nixdorf. Research projects conducted at the DFKI are funded by the German Ministry of Education, Science, Research and Technology, by the shareholder companies, or by other industrial contracts.

The DFKI conducts application-oriented basic research in the field of artificial intelligence and other related subfields of computer science. The overall goal is to construct systems with technical knowledge and common sense which - by using AI methods - implement a problem solution for a selected application area. Currently, there are the following research areas at the DFKI:

- ☐ Intelligent Engineering Systems
- ☐ Intelligent User Interfaces
- ☐ Computer Linguistics
- ☐ Programming Systems
- ☐ Deduction and Multiagent Systems
- ☐ Document Analysis and Office Automation.

The DFKI strives at making its research results available to the scientific community. There exist many contacts to domestic and foreign research institutions, both in academy and industry. The DFKI hosts technology transfer workshops for shareholders and other interested groups in order to inform about the current state of research.

From its beginning, the DFKI has provided an attractive working environment for AI researchers from Germany and from all over the world. The goal is to have a staff of about 100 researchers at the end of the building-up phase.

Dr. Dr. D. Ruland
Director

Best-First Surface Realization

Stephan Busemann

DFKI-RR-96-05

Appears in Proc. 8th International Workshop on Natural Language Generation, Herstmonceux, Sussex, Great Britain, June 1996

This work has been supported by a grant from The Federal Ministry of Education, Science, Research and Technology (FKZ ITWM-9402).

ISSN 0946-008X

Best-First Surface Realization

Stephan Busemann
email: `busemann@dfki.uni-sb.de`

Abstract

Current work in surface realization concentrates on the use of general, abstract algorithms that interpret large, reversible grammars. Only little attention has been paid so far to the many small and simple applications that require coverage of a small sublanguage at different degrees of sophistication. The system TG/2 described in this paper[1] can be smoothly integrated with deep generation processes, it integrates canned text, templates, and context-free rules into a single formalism, it allows for both textual and tabular output, and it can be parameterized according to linguistic preferences. These features are based on suitably restricted production system techniques and on a generic backtracking regime.

1 Motivation

Current work in surface realization concentrates on the use of general, abstract algorithms that interpret declaratively defined, non-directional grammars. It is claimed that this way, a grammar can be reused for parsing and generation, or a generator can interpret different grammars (e.g. in machine translation). A prominent example for this type of abstract algorithm is semantic-head-driven generation [Shieber *et al.*, 1990] that has been used with HPSG, CUG, DCG and several other formalisms.

In practice, this type of surface realization has several drawbacks. First, many existing grammars have been developed with parsing as the primary type of processing in mind. Adapting their semantics layer to a generation algorithm, and thus achieving reversibility, can turn out to be a difficult enterprise [Russell *et al.*, 1990]. Second, many linguistically motivated grammars do not cover common means of information presentation, such as filling in a table, bulletized lists, or semi-frozen formulae used for greetings in letters. Finally, the grammar-based logical form representation hardly serves as a suitable interface to deep generation processes. Grammar-based semantics is, to a large extent, a compositional reflex of the syntactic structure and hence corresponds too closely to the surface form to be generated. As a consequence, only little attention has been paid to interfacing this type of realizers adequately to deep generation processes, e.g. by allowing the latter to influence the order of results of the former.

The system TG/2, which is presented in this contribution, overcomes many flaws of grammar-based surface realization systems that arise in concrete applications. In particular, TG/2

---

[1] I am grateful to Michael Wein, who implemented the interpreter, and to Jan Alexandersson for influential work on a previous version of the system. Finally, I wish to thank two anonymous reviewers for useful suggestions. All errors contained in this paper are my own.

- can be smoothly integrated with 'deep' generation processes,
- integrates canned text, templates, and context-free rules into a single formalism,
- allows for both textual and tabular output,
- efficiently reuses generated substrings for additional solutions, and
- can be parameterized according to linguistic properties (regarding style, grammar, fine-grained rhetorics etc.).

TG/2 is based on restricted production system techniques that preserve modularity of processing and linguistic knowledge, hence making the system transparent and reusable for various applications. Production systems have been used both for modeling human thought (e.g. [Newell, 1973]) and for the construction of knowledge-based expert systems (e.g. [Shortliffe, 1976]). In spite of the modularity gained by separating the rule basis from the interpreter, production systems have disappeared from the focus of current research because of their limited transparency caused by various types of side effects. In particular, side effects could modify the data base in such a way that other rules become applicable [Davis and King, 1977].

However, precondition-action pairs can be used in a more restricted way, preserving transparency by disallowing side effects that affect the database. In TG/2 preconditions are tests over the database contents (the generator's input structure), and actions typically lead to a new subset of rules the applicability of which would be tested on some selected portion of the database. By constraining the effects of production rules in such a way, the disadvantages of early production systems are avoided. At the same time, considerable flexibility is maintained with regard to linguistic knowledge used. A production rule may

- involve a direct mapping to surface forms (canned text),
- require to fill in some missing portion from a surface text (template), or
- induce the application of other rules (classical grammar rules)

Early template-based generation methods have correctly been criticized for beeing too inflexible to account adequately for the communicative and rhetorical demands of many applications. On the other hand, templates have been successfully used when these demands could be hard-wired into the rules. In TG/2 the rule writer can choose her degree of abstraction according to the task at hand. She can freely intermix all kinds of rules.

The rest of the paper is organized as follows. TG/2 assumes as its input a predicate-argument structure, but does not require any particular format. Rather, a separate translation step is included that translates the output of feeding components into expressions of the Generator Interface Language (GIL) (Section 2). In Section 3 the formalism TGL (Template Generation Language) for production rules is introduced. The properties of TGL allow for efficient generation of all possible solutions in any order. The TGL interpreter and its generic backtracking regime are presented in Section 4. It is used to parameterize TG/2 by inducing an order in which the solutions are generated (Section 5).

Figure 1 gives an overview of the system and its components.

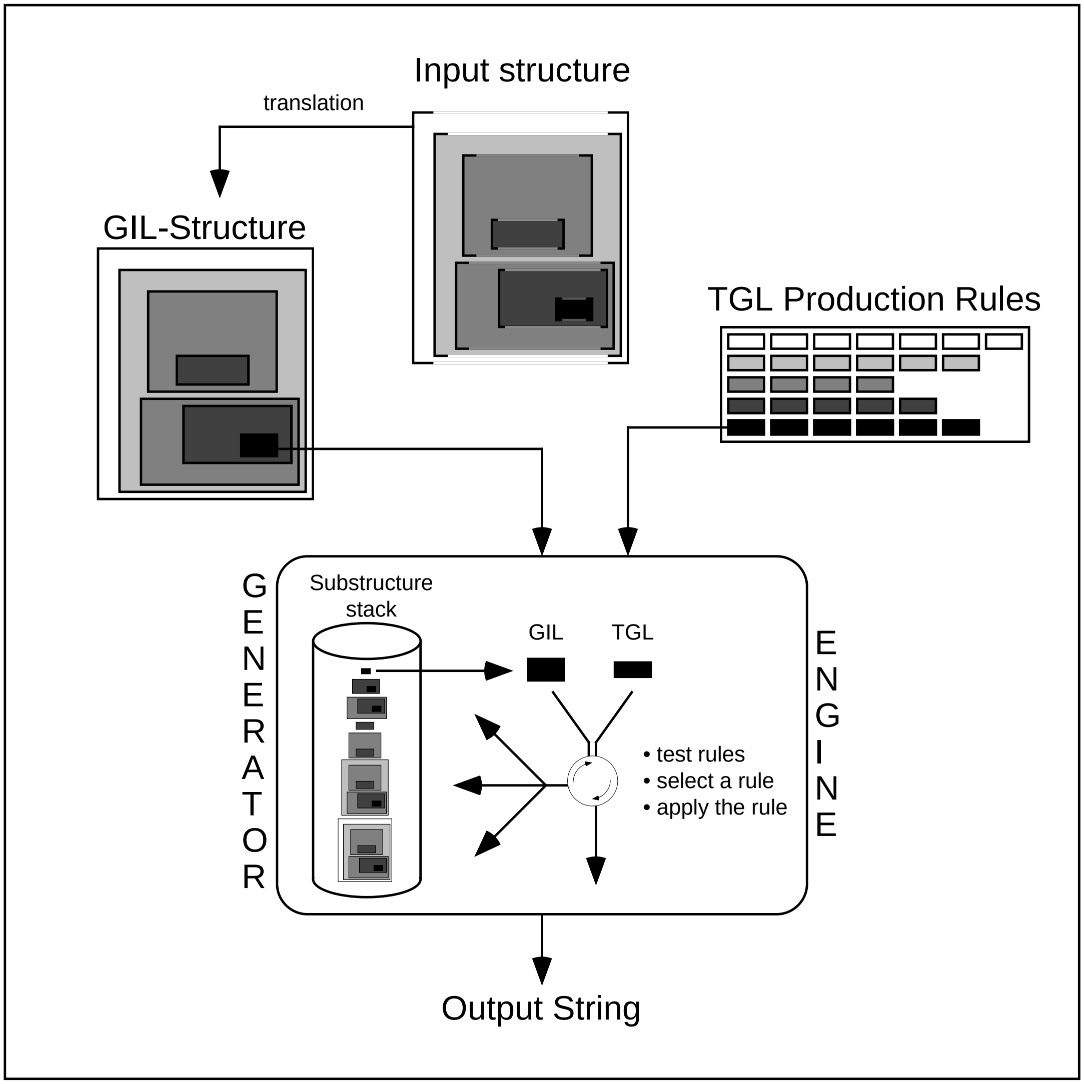

Figure 1: Overview of the system TG/2.

2 The Generation Interface Language (GIL)

Although the level of logical form is considered a good candidate for an interface to surface realization, practice shows that notational idosyncrasies can pose severe translation problems. TG/2 has an internal language, GIL, that corresponds to an extended predicate argument structure. GIL is the basis for the precondition test predicates and the selector functions of TGL. Any input to TG/2 is first translated into GIL before being processed. It is of considerable practical benefit to keep the rule basis as independent as possible from external conditions (such as changes to the output specification of the feeding system).

GIL is designed to be a target language suited for deep generation processes. Similar aims have been pursued with the development of the Sentence Plan Language (SPL) [Kasper and Whitney, 1989] that is used in a variety of generation systems. Like SPL, GIL assumes only little grammatical information. GIL can represent DAG-like feature structures. Figure 2

```
[(PRED request)
 (HEARER [(ID refo365) (SET < nussbaum >)])
 (SPEAKER [(ID refo752) (SET < digisec >)])
 (THEME [(SMOOD [(TOPIC #1) (MODALITY unmarked) (TIME pres)])
         (PRED meet)
         (DREF [(ID refo610) (SET < meet1 >)])
         (ARGS < #1= [(ROLE agent)
                      (CARD single)
                      (CONTENT [(DREF [(ID refo621) (SET < zweig >)])
                                (QFORCE noquant)
                                (PRED humname)
                                (NAME [(TITLE \"Prof.\")
                                       (SURNAME \"Zweig\")
                                       (SORT female)])])],
                 [(ROLE patient)
                  (CARD single)
                  (CONTENT [(DREF [(ID refo365) (SET < nussbaum >)])
                            (QFORCE iota)
                            (PRED object)])] >)
         (TIME-ADJ [(ROLE on) (CONTENT [(WEEKDAY 5)])])])]
```

Figure 2: A sample GIL input structure (*Prof. Zweig will Sie am Freitag treffen* [Prof. Zweig wants to meet you on Friday]. `<` and `>` are list delimiters; `#` denotes coreferences.

contains a sample GIL expression. The example shows the major language elements:

- The top level consists of a speech act predicate and arguments for author, addressee and theme (the speechact proper).
- Discourse objects can be assigned unique constants (`ID`) that denote `SET`s of discourse objects.
- `SMOOD` expresses sentence modalities including sentence type, time, a specification of which constituents to topicalize in a German declarative sentence, etc.
- The predicate argument structure is reflected by corresponding features; `ARGS` contains a list of arguments.
- Different sorts of free temporal and local adjuncts can be specified by corresponding features. In Figure 2, a temporal adjunct is represented under `TIME-ADJ`.
- Arguments and, in part, adjuncts are specified for their role, for cardinality, for quantificational force (under `CONTENT.QFORCE`), and further details such as name strings and natural gender.
- Temporal adjuncts relate to some context (e.g. *tomorrow*) or are indexical (e.g. *on Wednesday, February 7, 1996*). All common combinations in German are covered.

```
<rule>      ::= (DEFPRODUCTION <string> <tgl-rule>)

<tgl-rule>  ::= (:PRECOND (:CAT <category>
                         :TEST (<lisp-code>+))
                 :ACTIONS (:TEMPLATE <template>+
                          {:SIDE-EFFECTS <lisp-code>}
                          {:CONSTRAINT <feature-equation>+}))

<category>  ::= TXT | S | VP | NP | PP | PPdur | INF | ADJ | ...

<template>  ::= (:RULE <category> <lisp-code>) |
                (:OPTRULE <category> <lisp-code>) |
                (:FUN <lisp-code>) |
                <string>
```

Figure 3: An excerpt of TGL Syntax.

3 The Template Generation Language (TGL)

TGL defines a general format for expressing production rules as precondition-action pairs (cf. Figure 3). A TGL rule is applicable if its preconditions are met. A TGL rule is successfully applied, if the action part has been executed without failure. Failure to apply a TGL rule signals that the rule does not cover the portion of the input structure submitted to it.

Figure 4 shows a sample TGL rule. It corresponds to an infinitival VP covering a direct object, an optional temporal adjunct, an optional expression for a duration (such as *for an hour*), an optional local adjunct (such as *at the DFKI building*), and the infinite verb form. Given the input GIL structure of Figure 2, the VP *Sie am Freitag treffen* [to meet you on Friday] could be generated from this rule. Among the optional constituents, only the temporal adjunct would find appropriate material in the GIL input structure (under `THEME.TIME-ADJ`).

Every TGL rule has a unique name, denoted by the initial string. This name is used for expressing preferences on alternative rules (cf. Section 5).

Category: The categories can be defined as in a context-free grammar. Correspondingly, categories are used for rule selection (see below). They ensure that a set of TGL rules possesses a context-free backbone.

Test: The Lisp code under `:TEST` is a boolean predicate (usually about properties of the portion of input structure under investigation or about the state of some memory). In the sample rule, an argument is required that fills the patient role.

Template: Actions under `:TEMPLATE`[2] include the selection of other rules (`:RULE, :OPTRULE`), executing a function (`:FUN`), or returning an ASCII string as a (partial) result.

When selecting other rules by virtue of a category, a Lisp function is called that identifies the relevant portion of the input structure for which a candidate rule must pass its associated tests. In Figure 4, the first action selects all rules with category `NP`; the

---

[2] The notion of template is preserved for historical reasons: the predecessor system TG/1 was strictly template-based.

```
(defproduction "VPinf with temp/loc adjuncts"
        (:PRECOND (:CAT VP
                   :TEST ((role-filler-p 'patient)))
         :ACTIONS (:TEMPLATE (:RULE NP (role-filler 'patient))
                             (:OPTRULE PP (temp-adjunct))
                             (:OPTRULE PPdur (temp-duration))
                             (:OPTRULE PP (loc-adjunct))
                             (:RULE INF (theme))
                   :CONSTRAINTS (CASE (NP) :VAL 'akk))))
```

Figure 4: A sample production rule for a VP with an infinitive verb form placed at the end.

relevant substructure is the argument filling the patient role (cf. the second element of the `ARGS` list in Figure 2). If there is no such substructure, an error is signalled[3] unless an `OPTRULE` slot (for "optional rule") was executed. In this case, processing continues without results from that slot.

Functions must return an ASCII string. They are mostly used for word inflection; otherwise, for German every inflectional variant would have to be encoded as a rule. TG/2 uses the morphological inflection component MORPHIX [Finkler and Neumann, 1988].

Side effects: The Lisp code under `:SIDE-EFFECTS` is a function whose value is ignored. It accounts for non-local dependencies between substructures, such as updates of a discourse memory. Note that these effects can be traced and undone in the case of backtracking.

Constraints: Agreement relations are encoded into the rules by virtue of a PATR style [Shieber *et al.*, 1983] feature percolation mechanism. The rules can be annotated by equations that either assert equality of a feature's value at two or more constituents or introduce a feature value at a constituent. Attempting to overwrite a feature specification yields an error. In Figure 4, the right-hand side constituent NP is assigned accusative case. Any of these effects are subject to backtracking.

Using TGL, small task- and domain-specific grammars can be written quickly. For instance, in the domain of appointment scheduling the system COSMA [Busemann *et al.*, 1994] has to accept, reject, modify, or refine suggested meeting dates via email. The sublanguage encoded in TGL only needs a few speech acts, about twenty sentential templates, and a complete account of German date expressions. Moreover, formal as well as informal opening and closing phrases for emails are covered.

Larger grammars may become difficult to maintain unless special care is taken by the grammar writer to preserve a global structure of rules, both by defining suitable categories and by documenting the rules. TGL rules are presently written using a text editor. A specialized TGL grammar editor could improve the development and the organization of grammars considerably. Syntactic correctness is checked at compile-time by an LR-Parser generated by Zebu [Laubsch, 1992] on the basis of a BNF syntax for TGL.

---

[3] In the case at hand, the grammar writer preferred to ensure availability of the substructure by virtue of the test predicate.

4 An interpreter with generic backtracking

TG/2 has a simple interpretation procedure that corresponds to the classical three-step evaluation cycle in production systems (matching, conflict resolution, firing) [Davis and King, 1977]. The algorithm receives a GIL structure as its input and uses a distinguished category, `TXT`, to start from.

1. **Matching:** Select all rules carrying the current category. Execute the tests for each of these rules on the input structure and add those passing their test to the *conflict set.*
2. **Conflict resolution:** Select an element from the conflict set.
3. **Firing:** Execute its side effect code (if any). Evaluate its constraints (if any). For each action part, read the category, determine the substructure of the input by evaluating the associated function, and goto 1.

The processing strategy is top-down and depth-first. The set of actions is fired from left to right. Failure of executing some action causes the rule to be backtracked.

The interpreter yields all solutions the grammar can generate. It attempts to generate and output a first solution, producing possible alternatives only on external demand. Any alternative is based on backtracking at least one rule. Backtrack points correspond to conflict sets containing more than one element.

Backtracking may turn out to be inefficient if it involves recomputation of previously generated substrings. In TG/2 this effort is reduced considerably because it is only necessary to recompute the part licensed by the newly selected rule. What has been generated before or after it remains constant (modulo some word forms that need to agree with new material) and can thus be reused for subsequent solutions. This is possible due to the design properties of TGL: rules cannot irrevocably influence other parts of the solution. In particular, the context-free backbone implicit in any solution and the restrictions to side effects mentioned above keep the structural effects of TGL rules local.

In the sequel, technical aspects of the backtracking regime are discussed. Let us assume that the interpreter compute a backtrack point. Let us call the sequence of strings generated by previous actions its *pre-context*, the set of string sequences generated from the elements of the conflict set its *ego*, and the sequence of strings generated from subsequent actions its *post-context.* For every ego, the pre- or the post context may be empty.

Each time a backtrack point is encountered during processing, an entry into a global table is made by specifying its pre-context (which is already known due to the left-to-right processing), a variable for the ego (which will collect the sequences of strings generated by the elements of the conflict set), and a variable for the post-context (which is unknown so far).[4] Figure 5 shows the state of a sample table comprising three backtrack points after all solutions have been computed. The ego variable is shown using indices running over the elements of the respective conflict sets. The operator '$\cdot$' denotes concatenation of strings with strings or sets of strings, delivering all possible combinations.

After the first solution has been found (i.e. $s_1 \cdot s_{21} \cdot s_3 \cdot s_{51} \cdot s_{61} \cdot s_{71} \cdot s_8$), every ego set contains one element. The post contexts for all backtrack points can be entered into the table.

---

[4] In fact, it is preterminal rather than terminal elements that are stored in the table in order to account for modified constraints. This can be neglected in the present discussion, but will be taken up again below.

	pre context	*ego*	*post context*
B_1	s_1	$V_1 = \{s_{2i} \mid 1 \leq i \leq \lvert B_1 \rvert\}$	$s_3{\cdot}V_2{\cdot}s_8$
B_2	$s_1{\cdot}V_1{\cdot}s_3$	$V_2 = \{s_{4j} \mid 1 \leq j \leq \lvert B_2 \rvert\}$	s_8
B_{2_1}	$s_1{\cdot}V_1{\cdot}s_3{\cdot}s_{5j}$	$V_{2_1} = \{s_{6k} \mid 1 \leq k \leq \lvert B_{2_1} \rvert\}$ where $s_{4j} = s_{5j}{\cdot}V_{2_1}{\cdot}s_{7j}$	$s_{7j}{\cdot}s_8$

Figure 5: Table of Backtrack Points: B_2 is encountered outside of the ego of B_1. B_{2_1} is encountered inside the ego of B_2.

The next solution is generated by selecting anyone of the backtrack points and adding a new element to the ego set. At the same time, all other entries of the table are updated, and the set of additional solutions can be read off straightforwardly from the entry of the backtrack point just processed. Assume, for instance, that B_{2_1} generates a second solution, thus causing V_{2_1} to have two elements. We then get $s_1{\cdot}s_{21}{\cdot}s_3{\cdot}s_{51}{\cdot}s_{62}{\cdot}s_{71}{\cdot}s_8$. Now assume that B_1 also generates a second solution. This directly yields two more solutions since the post context of B_1 includes, via s_{4j}, the two elements of V_{2_1}.

This way only the alternative elements of a conflict set have to be expanded from scratch. All other material can be reused. This is highly efficient for backtrack points introducing "cheap" alternatives (e.g. different wordings). Since the ego must be recomputed from scratch, much less is gained with backtrack points occurring at a higher level (e.g. active vs. passive sentence). In order to avoid having to recompute successfully generated partial results within the ego, such results are stored during processing together with the part of the input structure and the current category. They can be reused when passing an applicability test that requires the stored category and input structure to be identical to the current ones.

The backtracking approach described is based on the assumption that any constraints introduced for some ego can be undone and recomputed on the basis of rules generating an alternative ego. Clearly, features instantiated for some ego may have effects onto the pre- or post-context. If an agreement feature receives a different value during backtracking and it relates to material outside the ego, inflectional processes for that material must be computed again. These cases can be detected by maintaining a trace of all constraint actions. The recomputation is rendered possible by adding, in addition to storing terminal strings in the table, the underlying calls to the inflection component as well.

5 Parameterization

Parameterization of TG/2 is based on specifying the way how the generic backtracking regime should operate. It can be influenced with regard to

- the element in the conflict set to be processed next, and
- the backtrack point to be processed next.

Both possibilities taken together allow a system that feeds TG/2 to specify linguistic criteria of preferred solutions to be generated first.

The criteria are defined in terms of rule names, and a criterion is fulfilled if some corresponding rule is successfully applied. We call such a rule *c-rule*. TG/2 implements a simple strategy that processes those backtrack points first that have conflict sets containing c-rules, and preferrably choses a c-rule from a conflict set. When applied incrementally, this procedure yields all solutions fulfilling (some of) the criteria first.

It would be desirable to see the solution fulfilling most criteria first. However, incremental application enforces decisions to be taken locally for each conflict set. Any c-rule chosen may be the last one in a derivation, whereas chosing a non-c-rule may open up further opportunities of chosing c-rules. These limits are due to a lack of look-ahead information: it is not known in general which decisions will have to be taken until all solutions have been generated.[5] Clearly, sacrificing incrementality is not what should be desired although it may be acceptable for some applications. The drawbacks include a loss of efficiency and run-time. This leaves us with two possible directions that can lead to improved results.

Analyzing dependencies of criteria: The solution fulfilling most criteria is generated first if sets of mutually independent criteria are applied: fulfilling one criterion must not exclude the applicability of another one, unless two criteria correspond to rules of the same conflict set. In this case, they must allow for the the application of the same subset of criteria. If these conditions are met, chosing a c-rule from every conflict set, if possible, will lead to a globally best solution first. There is, however, the practical problem that the conditions on the criteria can only be fulfilled by analyzing, and possibly modifying, the TGL grammar used. This contradicts the idea of having the user specify her preferences independent of TG/2 properties.

Learning dependencies of criteria: Missing look-ahead information could be acquired automatically by exploiting the derivational history of previously generated texts. For every applied rule, the set of c-rules applied later in the current subtree of a derivation is stored. From this information, we can derive off-line for any set of criteria which c-rules have applied in the corpus and how often each c-rule has applied within a derivation. Computing such information from the context-free backbone of TGL grammars instead would be less effective since it neglects the drastic filtering effects of preconditions. However, checking the grammar this way indicates which c-rules will *not* appear in some subtree.

During processing, TG/2 can then judge the global impact of chosing the locally best c-rule and decide to fulfill or violate a criterion. The success of this method depends on how well the derivation under construction fits with the sample data. The more examples the system observes, the more reliable will be its decisions.

The latter approach is in fact independent on how the criteria influence each other. In addition, it can be extended to cope with *weighted* criteria. A weight is specified by the user (e.g. a feeding system) and expresses the relative importance of the criterion being fulfilled in a solution. TG/2 would give preference to derivations leading to the maximum global weight. The global weight of a solution is the sum of the c-rule weights, each divided by the number of times the c-rule occurs.

However, different GIL structures may, for a TGL rule, lead to different sets of follow-up c-rules. This causes the decision to be nondeterministic unless the reasons for the difference are learned and applied to the case at hand. We must leave it to future research to identify and apply suitable learning algorithms to solving this problem.

Criteria have been implemented for choosing a language, for chosing between active and passive sentences, for preferring paratactical over hypotactical style, and for choice of formal vs. informal wordings. Additional uses could include some rhetorical structuring (e.g. order

---

[5] Note that this conclusion does not depend on the processing strategy chosen.

of nucleus and satellites in RST-based analyses [Mann and Thompson, 1988]).

The approach presented offers a technical framework that allows a deep generation process to abstract away from many idiosyncrasies of linguistic knowledge by virtue of meaningful weighting functions. Ideally, these functions must implement a theory of how mutual dependencies of criteria should be dealt with. For instance, lexical choice and constituent order constraints may suggest the use of passive voice (cf. e.g. [Danlos, 1987]). It is a yet open question whether such a theory can be encoded by weights. However, for some sets of preferences, this approach has proven to be sufficient and very useful.

6 Conclusion

In this contribution, we have introduced TG/2, a production-rule based surface generator that can be parameterized to generate the best solutions first. The rules are encoded in TGL, a language that allows the definition of canned text items, templates, and context-free rules within the same formalism. TGL rules can, and should, be written with generation in mind, i.e. the goal of reversibility of grammars pursued with many constraint-based approaches has been sacrificed. This is justified because of the limited usefulness of large reversible grammars for generation.

TGL is particularly well suited for the description of limited sublanguages specific to the domains and the tasks at hand. Partial reuse of such descriptions depends on whether the grammar writer keeps general, reusable definitions independent from the specific, non-reusable parts of the grammar. For instance, time and date descriptions encoded for the COSMA domain can be reused in other TG/2 applications. On the other hand, TGL sublanguage grammars can be developed using existing resources. For instance, suitable fragments of context-free grammars translated into TGL could be augmented by the domain and task specific properties needed. Practical experience must show whether this approach saves effort.

The system is fully implemented in Allegro Common Lisp and runs on different platforms (SUN workstations, PC, Macintosh). Computing the first solution of average-length sentences (10–20 words) takes between one and three seconds on a SUN SS 20. TG/2 is being used in the domain of appointment scheduling within DFKI's COSMA system. In the near future, the system will be used within an NL-based information kiosk, where information about environmental data must be provided in both German and French language, including tabular presentations if measurements of several substances are involved.

References

[Busemann *et al.*, 1994] S. Busemann, S. Oepen, E. Hinkelman, G. Neumann, and H. Uszkoreit. COSMA–multi-participant NL interaction for appointment scheduling. Research Report RR-94-34, DFKI, Saarbrücken, 1994.

[Danlos, 1987] L. Danlos. *The Linguistic Basis of Text Generation.* Cambridge University Press, Cambridge, 1987.

[Davis and King, 1977] R. Davis and J. King. An overview of production systems. In E. W. Elcock and D. Michie, editors, *Machine Intelligence 8*, pages 300–332. Ellis Horwood, Chichester, 1977.

[Finkler and Neumann, 1988] W. Finkler and G. Neumann. Morphix: A fast realization of a classification–based approach to morphology. In H. Trost, editor, *Proc. der 4. Österreichischen Artificial–Intelligence Tagung*, pages 11–19, Berlin, August 1988. Springer.

[Kasper and Whitney, 1989] R. Kasper and R. Whitney. SPL: A sentence plan language for text generation. Technical report, USC-ISI, Marina del Rey, 1989.

[Laubsch, 1992] J. Laubsch. Zebu: A Tool for Specifying Reversible LALR(1) Parsers. Technical Report HPL-92-147, Hewlett-Packard Labs, Palo Alto, CA, July 1992.

[Mann and Thompson, 1988] W. C. Mann and S. A. Thompson. Rhetorical structure theory: Toward a functional theory of text organization. *Text*, 8(3):243–281, 1988.

[Newell, 1973] A. Newell. Production systems: Models of control structures. In W. G. Chase, editor, *Visual Information Processing*, pages 463–526. Academic Press, New York, 1973.

[Russell *et al.*, 1990] G. Russell, S. Warwick, and J. Carroll. Asymmetry in parsing and generating with unification grammars: Case studies from ELU. In *Proc. 28th ACL*, pages 205–211., Pittsburgh, 1990.

[Shieber *et al.*, 1983] S. Shieber, H. Uszkoreit, F. Pereira, J. Robinson, and M. Tyson. The formalism and implementation of PATR-II. In B. J. Grosz and M. E. Stickel, editors, *Research on Interactive Acquisition and Use of Knowledge*, pages 39–79. AI Center, SRI International, Menlo Park, CA, 1983.

[Shieber *et al.*, 1990] S. Shieber, G. van Noord, R. C. Moore, and F. Pereira. A semantic-head-driven generation algorithm for unification-based formalisms. *Computational Linguistics*, 16(1):30–42, 1990.

[Shortliffe, 1976] E. H. Shortliffe. *Computer-based Medical Consultations: MYCIN*. Elsevier, New York, 1976.