Reversible Unification Based Machine Translation

Gertjan van Noord
OTS RUU Trans 10
3512 JK Utrecht
vannoord@hutruu59.BITnet

November 25, 1994

Abstract

In this paper it will be shown how unification grammars can be used to build a *reversible* machine translation system.

Unification grammars are often used to define the relation between strings and meaning representations in a declarative way. Such grammars are sometimes used in a bidirectional way, thus the same grammar is used for both parsing and generation. In this paper I will show how to use bidirectional unification grammars to define *reversible* relations between language dependent meaning representations. Furthermore it is shown how to obtain a completely reversible MT system using a series of (bidirectional) unification grammars.

1 Introduction

The notion of a reversible MT system was first expressed by Landsbergen [11]. Such a system will in principle produce a *set of possible translations*, by employing linguistic knowledge only. Choosing the *best* translation from the set of linguistically possible translations will usually require other sources of knowledge, either incorporated in the system or provided (interactively) by the user. The relation 'possible translation' is symmetric whereas the relation 'best translation' is not. Thus an MT system may consist of a reversible core, implementing the symmetric relation 'possible translation', and additional components (not necessarily reversible) to select the best translation.

Not only is it *possible* to build reversible (modules of) MT systems; it has also been claimed that reversible systems are *preferable*. For example Isabelle [6] claims that reversible MT systems are to be preferred to others because in reversible MT systems a better understanding of the translation relation is achieved; such systems will eventually exhibit better practical performance. Moreover, the arguments in favour of using bidirectional grammars in NLP, such as those given in [1, 8] carry over to translation as well.

Because of the declarative nature of unification- and logic grammar formalisms grammars written in these formalisms are increasingly used in a bidirectional way, thus the same grammar is used for both parsing and generation. Some recent developments are reported in [3, 24, 16, 21, 2, 18, 19, 22, 20].

In this paper I will show how to use such bidirectional unification grammars to build a completely reversible, multilingual, MT system. For each language there is a unification grammar that defines a reversible relation between strings and language dependent meaning representations (*logical forms*). Moreover, for each language pair (or set of languages) there is a unification grammar that defines a reversible relation between such language dependent logical forms. Translation is thus defined by a series of three unification grammars.

A specific version of the system that is described here is implemented as the core of the experimental MiMo2 translation system [23]. This system aims at translating international news items on Teletext. Apart from unification grammars the system uses a bidirectional two-level orthography component. Language dependent meanings are represented as simple predicate argument structures with some extra labels indicating 'universal' meaning such as tense and aspect. The current system (November 1989) includes grammars for Dutch, Spanish and English.

The paper is set up as follows. In section 2, I will give some examples that show how bidirectional unification grammars can be used to define relations between logical forms of different languages. In section 3, reversibility is defined in terms of symmetry and computability. Possible approaches to obtain reversibility are discussed. In section 4, I will compare the current approach with some other approaches in the unification based translation paradigm and discuss some problems and future directions.

2 Unification-based Transfer

In this section I will give some examples of the use of a unification grammar (in PATR II [17] notation) to define the relation between language dependent logical forms. For illustrative purposes I will assume logical forms are represented by feature structures consisting of the attributes *pred*, $arg1$, $arg2$ together with some attributes representing 'universal' meanings such as tense, aspect, number and person; I will not touch upon issues such as quantification and modification. The logical forms of English and Spanish are labeled by the attributes *gb* and *sp* respectively. As an example the logical form of 'The army opened fire at the civilians' is represented as in figure 1. Such feature struc-

Figure 1: An example of a logical form

$$\left[\begin{array}{l} gb = \left[\begin{array}{l} pred = open\_fire\_at \\ arg1 = \left[\begin{array}{l} pred = army \\ number = sg \end{array}\right] \\ arg2 = \left[\begin{array}{l} pred = civilian \\ number = pl \end{array}\right] \end{array}\right] \end{array}\right]$$

tures will often be related in a straightforward way to a Spanish equivalent, except for the value of the *pred* attributes. A very simple rule in PATR II style may look as in figure 2. This rule simply states that the

Figure 2: A simple rule

```
0 → 1 2 3
⟨0 gb pred⟩ = ⟨1 gb⟩
⟨0 gb arg1⟩ = ⟨2 gb⟩
⟨0 gb arg2⟩ = ⟨3 gb⟩
⟨0 sp pred⟩ = ⟨1 sp⟩
⟨0 sp arg1⟩ = ⟨2 sp⟩
⟨0 sp arg2⟩ = ⟨3 sp⟩
```

translation of a logical form is composed of the translation of its arguments. If the rule applies to the feature structure in 1 the three daughters of the rule will be instantiated as in figure 3, and the value of *sp* will be bound to the *sp* values of these daughters. An example

Figure 3: Three instantiations

$$\left[\; gb = open\_fire\_at \;\right]$$

$$\left[\; gb = \left[\begin{array}{l} pred = civilian \\ number = pl \end{array}\right] \right]$$

$$\left[\; gb = \left[\begin{array}{l} pred = army \\ number = sg \end{array}\right] \right]$$

of the rule for the first daughter will be a lexical entry and looks as in figure 4. The simple English expression 'army' has to be translated as a complex expression in Spanish: 'fuerza militar'. The rule will look as in 5 where it is assumed that the construction is analyzed in Spanish as an ordinary noun-adjective construction, and where the logical form of the adjective takes the logical form of the noun as its argument. The translation for 'civilian' is defined in a similar rule (although the translation of 'number' is different). Note that this example of complex transfer is similar to the famous 'schimmel - grey horse' cases. As a result of the rule applications the feature structure in figure 1 will get

Figure 4: A lexical entry

```
0 →
⟨0 gb⟩ = open_fire_at
⟨0 sp⟩ = romper_el_fuego_a
```

Figure 5: A rule for 'fuerza militar'

```
0 →
⟨0 gb pred⟩ = army
⟨0 sp pred pred⟩ = militar
⟨0 sp arg1 pred⟩ = fuerza
⟨0 sp arg1 number⟩ = ⟨0 gb number⟩
```

instantiated to the feature structure in 6, from which the generator generates the string 'La fuerza militar rompio el fuego a la poblacion civil'.

Figure 6: The feature structure after transfer

$$\left[\begin{array}{l} gb = \left[\begin{array}{l} pred = open\_fire\_at \\ arg1 = \left[\begin{array}{l} pred = army \\ number = sg \end{array}\right] \\ arg2 = \left[\begin{array}{l} pred = civilian \\ number = pl \end{array}\right] \end{array}\right] \\ sp = \left[\begin{array}{l} pred = romper\_el\_fuego\_a \\ arg1 = \left[\begin{array}{l} pred = \left[\; pred = militar \;\right] \\ arg1 = \left[\begin{array}{l} pred = fuerza \\ number = sg \end{array}\right] \end{array}\right] \\ arg2 = \left[\begin{array}{l} pred = \left[\; pred = civil \;\right] \\ arg1 = \left[\begin{array}{l} pred = poblacion \\ number = sg \end{array}\right] \end{array}\right] \end{array}\right] \end{array}\right]$$

In the foregoing examples the relation between logical forms is rather straightforward. Note however that the full power of a unification grammar can be used to settle more difficult translation cases, because different attributes can be used to represent the 'translational syntax'. For instance we can build a tree as value of the attribute *tree* to represent the derivational history of the translation process. Or we can 'thread' information through different nodes to be able to make translations dependent on each other. Translation parameters such as style and subject field can be percolated as attributes of nodes to obtain consistent translations; but these attributes themselves need not be translated.

3 Reversible Unification Grammars

A unification grammar defined in formalisms such as PATR II and DCG [12] usually defines a relation between a string of words and a logical form. In sign-based approaches such as UCG [26] and HPSG [14] this string of words is not assigned a privileged status but is the value of one of the attributes of a feature structure. I will assume a formalism similar to PATR II, but without the context-free base; the string is represented as the value of one of the attributes of a feature structure. Thus more generally, unification grammars define relations between the values of two (or more[1]) attributes - for example the relation between the value of the attributes *string* and lf, or between the value of the attributes sp and gb; these relations are all relations between feature structures.

3.1 Reversibility

I will call a binary relation *reversible* if the relation is *symmetric* and *computable.* Both symmetry and computability will be explained in the following subsections. A grammar G is reversible for a relation R iff R is reversible and defined by G. For example, a grammar that relates strings to logical forms is reversible if both the parsing and generation problem is computable, and the relation between strings and logical forms is symmetric; the parsing problem is computable if for a given string all corresponding logical forms can be enumerated by some terminating procedure; such a procedure should halt if the given string does not have a corresponding logical form. Thus: reversible = symmetric + computable. Note that reversibility as defined here is different from bidirectionality. The latter merely says that grammars are to be used in two directions, but does not state how the two directions relate.

It is easy to see that a composition of reversible relations is a a reversible relation too; i.e. if some feature structure f_1 is related to some feature structure f_n via the reversible relations $R_i(f_i, f_{i+1})$, each defined by some reversible grammar G_i, then $R'(f_1, f_n)$ is reversible. Thus an MT system that defines a relation $R(s_s, s_t)$ via the relations $R_1(s_s, l_s)$, $R_2(l_s, l_t)$ and $R_3(l_t, s_t)$ is reversible if $R_{1,2,3}$ are reversible.

3.1.1 Symmetry

A relation $R \subseteq A \times B$ is symmetric iff $R(a, b)$ implies $R(b, a')$ where a and a' are equivalent. For an MT system we want to define 'equivalence' in such a way that the translation relation is a symmetric relation between strings. However, strings are feature structures thus we must define equivalence for feature structures to obtain this effect.

Unification grammars as they are commonly used implement a rather *weak* notion of equivalence between feature structures: feature structures a and b are equivalent if they can unify:

Definition 1 (Weak equivalence) *Two feature structures f_1, f_2 are weakly equivalent iff $f_1 \sqcup f_2$ exists.*

If feature structures are taken to stand for all their ground instances this yields an acceptable version of symmetry. Moreover, under the assumption that feature structures which represent strings are always ground (i.e. these feature structures cannot be extended), this results in a symmetric relation between (feature structures that represent) strings.

It is also possible to define a 'strong' notion of equivalence for feature structures that does not rely on this assumption.

Definition 2 (Strong equivalence) *Two feature structures f_1, f_2 are strongly equivalent ($f_1 \equiv f_2$) iff $f_2 \sqsubseteq f_1$ and $f_1 \sqsubseteq f_2$.*

A grammar that defines a computable relation between two attributes under the strong definition of equivalence might be called *strongly reversible.* Similarly a *weakly reversible* grammar is reversible under a weak definition of equivalence. Again these results can be generalized to a series of unification grammars. The strong version of equivalence can be motivated on the ground that it may be easier to obtain computability; this is the topic of the next subsection. In section 3.2 I will discuss possible relaxations of the strong version of equivalence to obtain 'mildly' reversible grammars.

3.1.2 Computability

A relation $R \subseteq A \times B$ is *computable* iff for a given $a \in A$ the set $\{b \in B | R(a, b)\}$ can be enumerated by some terminating procedure. To discuss computability it is useful to look a bit more careful at the relations we are interested in. These relations are all binary relations between feature structures. However, in the case of the relation between strings and logical forms, strings will always be related to logical forms and logical forms will be related to strings. Similarly for the relation between Dutch and Spanish logical forms. Clearly, the domain and range of the relation is structured and can be partioned into two sets A and B, for example the set of feature structures representing strings and the set of feature structures representing logical forms. The relation $R \subseteq A \cup B \times A \cup B$ can be partitioned similarly into the relations $r \subseteq A \times B$ and its inverse, $r^{-1} \subseteq B \times A$. The problem to compute R is now replaced by two problems: the computation of r and r^{-1}. For example the problem to compute the relation between logical forms and strings consists of the parsing- and generation problem. It is now possible to incorporate the notion of equivalence, to obtain a definition of a parser, generator and translator. For example, an algorithm that computes the foregoing relation r will enumerate for a given features structure f_1 all feature structures f_2, such that $r(f_3, f_2)$ and f_1 and f_3 are equivalent. In the case of strong equivalence this implies that $f_1 \sqsubseteq f_3$ (*completeness*), and $f_3 \sqsubseteq f_1$ (*coherence*). In other words, the input should not be extended (coherence) and should completely be derived (completeness). This usage of the terms completeness and coherence was introduced in [24]. In the following I will discuss ways to obtain computability of one such partition.

It is well known that relations defined by unrestricted unification grammars are not computable in general as such grammars have Turing power [13]; it is thus not

[1] Note that it is possible to define a unification grammar that relates several language dependent logical forms; in this approach a multilingual transfer system consists of only one transfer grammar.

decidable whether the relation is defined for some given input. Usually some constraint on grammars is defined to remedy this. For example the off-line-parsability constraint [13, 5] ensures that the recognition problem is solvable. Moreover this constraint also implies that the parsing problem as defined here is computable; i.e. the proof procedure will always terminate (because the constraint implies that there is a limit to the depth of possible parse trees for all strings of a given length).

However the off-line-parsability constraint assumes a context-free base of the formalism. A generalization of the off-line-parsability constraint for any binary relation defined by unification grammars will consist of three parts; the first and third of these parts are usually implicit in the case of parsing.

First, the value of the input must be built in a well-behaved compositional way. For example in the case of parsing: each daughter of a rule dominates part of the string dominated by the mother of that rule. Similarly for transfer and generation: each daughter of a rule has a value for lf that is part of the value of lf of the mother.

Second, a special condition is defined for rules where the input value of the mother is the same as the input value of one of the daughters. For parsing such rules have exactly one daughter. A chain of applications of such rules is disallowed by some constraint or other; this is the core of most definitions of the off-line parsability constraint. For example in [13] such a chain is disallowed as the principal functor of a term may only occur once in a chain. For a slightly more general definition, cf. [5]. For generation and transfer a similar constraint can be defined. In the terminology of [18, 19] the 'head' of a rule is a daughter with the same logical form as its mother. A chain of these heads must be disallowed.

Third, the input should not get extended during the proof procedure. In the case of parsing this is achieved easily because the input is ground [2]. For generation and transfer this is not necessarily the case. This is the point where the usefulness of the coherence condition comes in; the coherence requirement explicitly states that extension of the input is not allowed. For this reason strong reversiblity may be easier to obtain than weak reversibility. In the next subsection I will discuss two relaxations of strong symmetry that will not affect the computability properties discussed here.

Generalizing the terminology introduced by [13] a proof procedure is *strongly stable* iff it always terminates for grammars that adhere to a generalized off-line parsability constraint. In [15] a general proof procedure for DCG based on head-driven generation [18, 19, 22] is defined that is strongly stable for a specific instantiation of the generalized off-line parsability constraint.

[2]Note that this is the reason that most DCG parsers expect that the input value of the string has an atomic tail, i.e. $parse([john, kisses, mary], [])$ will work fine, but $parse([john, kisses, mary|X], X)$ will cause problems.

3.2 Possible relaxations

It is easy to see that the completeness and coherence requirements make life hard for the rulewriter as she/he needs to know exactly what the possible values of inputs are for some component. It is possible to relax the completeness and coherence requirement in two ways that will not affect the reversibility properties between strings. The usefulness of these relaxations depends on the analyses a user wishes to define.

3.2.1 Cyclic and non-cyclic attributes

The first relaxation assumes that there is a sort system defined for feature structures that makes it possible to make a distinction between *cyclic* and *non-cyclic* attributes (cf. [5]). For the moment a non-cyclic attribute may be defined as an attribute with a finite number of possible values (i.e. it is not recursive). For example the attributes $arg1$ and $arg2$ will be cyclic whereas *number* will be non-cyclic. The completeness and coherence condition is restricted to cyclic attributes. As the proof procedure can only further instantiate non-cyclic attributes no termination problems occur because there are only a finite number of possibilities to do this. The definition of 'equivalence' for feature structures is now slightly changed. To define this properly it is necessary to define the notion *non-cyclic extension.* A non-cyclic extension of a feature structure only instantiates non-cyclic attributes. This results in the following definition of equivalence:

Definition 3 (Non-cyclic equivalent) *Two feature structures f_1, f_2 are non-cyclic equivalent iff $f_1' \equiv f_2'$ where f_n' are non-cyclic extensions of f_n.*

It will be clear that the usefulness of this definition depends heavily on the style of grammar writing that is used. Note that it is of course also possible to declare for each non-cyclic attribute whether the completeness and coherence requirements hold.

3.2.2 Reentrancies

The second relaxation is not without ramifications for the organization of a transfer grammar. This relaxation has to do with reentrancies in feature structures. Some constructions such as control verbs and relative clauses may be represented using reentrancies; for example 'the soldiers tried to shoot the president' may be represented by a feature structure where the first argument of 'try' is reentrant with the first argument of 'shoot', cf. figure 7. The translation of such logical forms to Dutch equivalents can be defined as in rule 8. In this rule the reentrancy is explicitly mentioned for two reasons. The first reason is simply that in the case of different possible translations of $arg1$ we want the same translation for both $arg1$ and the embedded $arg1$. Note that the translation of 'soldier' into Dutch can be both 'soldaat' or 'militair'. If the reentrancy is not mentioned the system has to try to generate from four different Dutch logical forms, two of which without matching $arg1$'s.

Figure 7: A logical form containing reentrancy

$$\left[gb = \begin{bmatrix} pred = try \\ arg1 = \boxed{1} \begin{bmatrix} pred = soldier \\ number = pl \end{bmatrix} \\ arg2 = \begin{bmatrix} pred = shoot \\ arg1 = \boxed{1} \\ arg2 = \begin{bmatrix} pred = president \\ number = sg \end{bmatrix} \end{bmatrix} \end{bmatrix} \right]$$

Figure 8: Translating subject control

```
0 → 1 2 3
⟨0 gb pred⟩ = ⟨1 gb⟩
⟨0 nl pred⟩ = ⟨1 sp⟩
⟨0 gb arg1⟩ = ⟨0 gb arg2 arg1⟩
⟨0 nl arg1⟩ = ⟨0 nl arg2 arg1⟩
⟨0 gb arg1⟩ = ⟨2 gb⟩
⟨0 nl arg1⟩ = ⟨2 sp⟩
⟨0 gb arg2⟩ = ⟨3 gb⟩
⟨0 nl arg2⟩ = ⟨3 sp⟩
```

The reentrancy is also mentioned because this is required by the completeness condition. It is possible to relax the completeness and coherence condition with respect to these reentrancies, again without affecting the reversibility properties of the system by slightly changing the definition of equivalence. There is a trade-off between simplicity of the transfer grammar (in the presence of this relaxation) and the efficiency of the system. In the case of this relaxation the system will eventually find the good translations, but it may take a while. On the other hand, if we are to mention all (possibly unbounded) reentrancies explicitly then the transfer grammar will have to be complicated by a threading mechanism to derive such reentrancies. Again, the specific use of reentrancies in the logical forms that are defined will determine whether this relaxation is desired or not.

4 Final remarks

The objective to build a reversible MT system using a series of unification grammars is similar to the objective of the CRITTER system as expressed in [3, 7], and the work of Zajac in [25]. Instead of using unification grammars CRITTER uses logic grammars; Zajac uses a type system including an inheritance mechanism to define transfer-like rules. In these two approaches less attention is being paid to an exact definition of reversibility; although our work may be compatible with these approaches.

A somewhat different approach is advocated in [9]. In that approach a system is described where an LFG grammar for some source language is augmented with equations that define (part of) the target level representations. A generator derives from this partial description a string according to some LFG grammar of the target language. Instead of a series of three grammars this architecture thus assumes two grammars, one of which both defines the source language and the relation with the target language. The translation relation is not only defined between logical forms but may relate all levels of representation (*c-structure, f-structure, σ-structure*). Although in this approach monolingual grammars may be used in a bidirectional way it is unclear whether the translation equations can be used bidirectionally [3].

An important problem for the approach advocated here is the problem of *logical form equivalence*. Shieber [16] noted that unification grammars usually define a relation between strings and some *canonical* logical form. Depending on the nature of logical forms that are being used, several representations of a logical form may have the same 'meaning'; just as in first order predicate calculus the formulas $p \vee q$ and $q \vee p$ are logically equivalent; a unification grammar will not know of these equivalences and, consequently, all equivalences have to be defined separately (if such equivalents are thought of as being translational equivalents); for example in a transfer grammar two rules may be defined to translate $p \vee q$ into both $p' \vee q'$ and $q' \vee p'$ if these formulas are thought of as being equivalent. Of course this technique can only be applied if the number of equivalences is finite. It is not possible to define that p is equivalent with $\neg \ldots \neg p$ for any even number of $\neg$'s.

The approach discussed so far can be extended just as unification grammars for parsing and generation have been extended. Apart from equational constraints it will be useful to add others such as disjunction and negation. Moreover it seems useful to allow some version of universal constraints or some inheritance mechanism to be able to express generalizations and exceptions more easily.

Acknowledgements

I want to thank Joke Dorrepaal, Pim van der Eijk, Maria Florenza, Dirk Heylen, Steven Krauwer, Jan Landsbergen, Michael Moortgat, Herbert Ruessink and Louis des Tombe. I was supported by the European Community and the NBBI through the Eurotra project.

References

[1] Douglas E. Appelt. Bidirectional grammars and the design of natural language generation systems. In *Theoretical Issues in Natural Language Processing 3*, 1987.

[2] Jonathan Calder, Mike Reape, and Henk Zeevat. An algorithm for generation in unification categorial grammar. In *Fourth Conference of the European Chapter of the Association for Computational Linguistics*, 1989.

[3] Marc Dymetman and Pierre Isabelle. Reversible logic grammars for machine translation. In *Proceedings of the Second International Conference on Theoretical*

[3] Although parsing of LFG's is decidable no such result is available for generation; note furthermore that according to [9] extension is allowed during generation.

and Methodological issues in Machine Translation of Natural Languages, 1988.

[4] Barbara Grosz, Karen Sparck Jones, and Bonny Lynn Webber, editors. *Readings in Natural Language Processing*. Morgan Kaufmann, 1986.

[5] Andrew Haas. A generalization of the offline parsable grammars. In *27th Annual Meeting of the Association for Computational Linguistics*, 1989.

[6] Pierre Isabelle. Towards reversible MT systems. In *MT Summit II*, 1989.

[7] Pierre Isabelle, Marc Dymetman, and Elliott Macklovitch. CRITTER: a translation system for agricultural market reports. In *Proceedings of the 12th International Conference on Computational Linguistics*, Budapest, 1988.

[8] Paul S. Jacobs. Achieving bidirectionality. In *Proceedings of the 12th International Conference on Computational Linguistics*, 1988.

[9] Ronald Kaplan, Klaus Netter, Jürgen Wedekind, and Annie Zaenen. Translation by structural correspondences. In *Fourth Conference of the European Chapter of the Association for Computational Linguistics*, 1989.

[10] Margaret King, editor. *Machine Translation, the State of the Art*. Edinburgh University Press, 1987.

[11] Jan Landsbergen. Isomorphic grammars and their use in the Rosetta translation system, 1984. paper presented at the tutorial on Machine Translation, Lugano 1984, Also appears in [10].

[12] Fernando C.N. Pereira and David Warren. Definite clause grammars for language analysis - a survey of the formalism and a comparison with augmented transition networks. *Artificial Intelligence*, 13, 1980. reprinted in [4].

[13] Fernando C.N. Pereira and David Warren. Parsing as deduction. In *21st Annual Meeting of the Association for Computational Linguistics*, 1983.

[14] Carl Pollard and Ivan Sag. *Information Based Syntax and Semantics*. Center for the Study of Language and Information Stanford, 1987.

[15] Herbert Ruessink and Gertjan van Noord. Remarks on the bottom-up generation algorithm. Technical report, Department of Linguistics, OTS RUU Utrecht, 1989.

[16] Stuart M. Shieber. A uniform architecture for parsing and generation. In *Proceedings of the 12th International Conference on Computational Linguistics*, 1988.

[17] Stuart M. Shieber, Hans Uszkoreit, Fernando C.N. 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*. SRI report, 1983.

[18] Stuart M. Shieber, Gertjan van Noord, Robert C. Moore, and Fernando C.N. Pereira. A semantic-head-driven generation algorithm for unification based formalisms. In *27th Annual Meeting of the Association for Computational Linguistics*, 1989.

[19] Stuart M. Shieber, Gertjan van Noord, Robert C. Moore, and Fernando C.N. Pereira. Semantic-head-driven generation. *Computational Linguistics*, 1990. To appear.

[20] Tomek Strzalkowski. Automated inversion of a unification parser into a unification generator. Technical report, Courant Institute of Mathematical Sciences, New York University, 1989. technical report 465.

[21] Gertjan van Noord. BUG: A directed bottom-up generator for unification based formalisms. *Working Papers in Natural Language Processing, Katholieke Universiteit Leuven, Stichting Taaltechnologie Utrecht*, 4, 1989.

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

[23] Gertjan van Noord, Joke Dorrepaal, Louis des Tombe, and Pim van der Eijk. The MiMo2 research system. OTS RUU Utrecht.

[24] Jürgen Wedekind. Generation as structure driven derivation. In *Proceedings of the 12th International Conference on Computational Linguistics*, 1988.

[25] Rémi Zajac. A transfer model using a typed feature structure rewriting system with inheritence. In *27th Annual Meeting of the Association for Computational Linguistics*, 1989.

[26] Henk Zeevat, Ewan Klein, and Jo Calder. Unification categorial grammar. In Nicholas Haddock, Ewan Klein, and Glyn Morrill, editors, *Categorial Grammar, Unification Grammar and Parsing*. Centre for Cognitive Science, 1987. Volume 1 of Working Papers in Cognitive Science.