Computational Aspects of M-grammars

Joep Rous

Philips Research Laboratories, P.O. Box 80.000
5600 JA Eindhoven, The Netherlands
E-mail: rous@rosetta.prl.philips.nl (uucp)

ABSTRACT

In this paper M-grammars that are used in the Rosetta translation system will be looked at as the specification of attribute grammars. We will show that the attribute evaluation order is such that instead of the special-purpose parsing and generation algorithms introduced for M-grammars in Appelo et al.(1987), also Earley-like context-free parsing and ordinary generation strategies can be used. Furthermore, it is illustrated that the attribute grammar approach gives an insight into the weak generative capacity of M-grammars and into the computational complexity of the parsing and generation process. Finally, the attribute grammar approach will be used to reformulate the concept of isomorphic grammars.

M-grammars

In this section we will introduce, very globally, the grammars that are used in the Rosetta machine translation system which is being developed at Philips Research Laboratories in Eindhoven. The original Rosetta grammar formalism, called M-grammars, was a computational variant of Montague grammar. The formalism was introduced in Landsbergen(1981). Whereas rules in Montague grammar operate on strings, M-grammar rules (M-rules) operate on labelled ordered trees, called S-trees. The nodes of S-trees are labelled with syntactic categories and attribute-value pairs. Because of the reversibility of M-rules, it is possible to define two algorithms: M-Parser and M-Generator . The M-Parser algorithm starts with a surface structure in the form of an S-tree and breaks it down into basic expressions by recursive application of reversed M-rules. The result of the M-Parser algorithm is a syntactic derivation tree which reflects the history of the analysis process. The leaves of the derivation tree are names of basic expressions. The M-Generator algorithm generates a set of S-trees by bottom-up application of M-rules, the names of which are mentioned in a syntactic derivation tree. Analogous to Montague Grammar, with each M-rule a rule is associated which expresses its meaning. This allows for the transformation of a syntactic derivation tree into a semantic derivation tree by replacing the name of each M-rule by the name of the corresponding meaning rule. In Landsbergen (1982) it was shown that the formalism is very well fit to be used in an interlingual machine translation system in which semantic derivation trees make up the interlingua. In the analysis part of the translation system an S-tree of the source language is mapped onto a set of semantic derivation trees. Next, each semantic derivation tree is mapped onto a set of S-trees of the target language. In order to guarantee that for a sentence which can be analysed by means of the source language grammar a translation can always be generated using the target language grammar, source and target grammars in the Rosetta system are *attuned*. Grammars, attuned in the way described in Landsbergen (1982), are called *isomorphic*.

Appelo et al.(1987) introduces some extensions of the formalism, which make it possible to assign more structure to an M-grammar. The new formalism was called *controlled* M-grammars. In this new approach a grammar consists of a set of subgrammars. Each of the subgrammars contains a set of M-rules and a regular expression over the alphabet of rule names. The set of M-rules is subdivided into meaningful rules and transformations. Transformations have no semantic relevance and will therefore not occur in a derivation tree. The regular expression can be looked at as a prescription of the order in which the rules of the subgrammar have to be applied. Because of these changes in the formalism, new versions of the M-Parser and M-Generator algorithm were introduced which were able to deal with subgrammars. These algorithms, however, are complex and result in a rather cumbersome implementation. In this paper we will show that they can be replaced by normal context-free parse and generation algorithms if we interpret an M-grammar as the specification of an attribute grammar (Knuth (1968), Deransart et al.(1988)).

M-grammars as attribute grammars

The control expression which is used in the definition of a Rosetta subgrammar specifies a regular language over the alphabet of rule names. Another way to define such a language is by means of a regular grammar. Let control expression ce_i of subgrammar i define the regular language $\mathcal{L}(i)$. Then we can construct a minimal regular grammar rg_i which defines the same language. The grammar rg_i will have the following form:

- A set of non-terminals $N_i = \{I_i^0, \ldots, I_i^{M_i}\}$
- A set of terminals Σ_i. Σ_i is the smallest set such that there is a terminal $\bar{r} \in \Sigma_i$ for each M-rule r .
- Start symbol I_i^0

- A set of production rules P_i containing the following type of rules:
  - $I_i^j \rightarrow \bar{r} I_i^k$, where $\bar{r} \in \Sigma_i$
  - $I_i^j \rightarrow I_i^k$
  - $I_i^j \rightarrow \epsilon$.

We will use the regular grammar defined above as a starting point for the construction of an attributed subgrammar. An elegant view of attribute grammars can be found in Hemerik (1984). Hemerik defines an attribute grammar as a context free grammar with parametrized non-terminals and production rules. In general, non-terminals may have a number of parameters - *attributes* - associated with them. Production rules of an attribute grammar are pairs (rule form, rule condition). From a rule form, production rules can be obtained by means of substitution of values for the attribute variables that satisfy the rule condition. In the grammars presented in this paper, non-terminals have only one attribute of type S-tree. The attribute grammar rules that are used throughout this paper also have a very restricted form. A typical attribute grammar rule r with context free skeleton $A \rightarrow BC$ will look like:

$$\left[\begin{array}{l} A < o > \rightarrow B < p > C < q > \\ (o,(p,q)) \in \mathcal{R} \end{array}\right.$$

Here, $A < o > \rightarrow B < p > C < q >$ is the rule form, o, p, q are the attributes and $(o,(p,q)) \in \mathcal{R}$ is the rule condition. $\mathcal{R}$ defines a relation between the attributes at the left-hand side and the attributes at the right-hand side of the rule form.
For each subgrammar rg_i, $(1 < i \leq M)$ we will construct an attributed subgrammar ag_i. Each constructed attributed subgrammar ag_i will have a start symbol I_i^0. First, however, we define two new attributed subgrammars that have no direct relation with a subgrammar of a given M-grammar: the *start* subgrammar and the *terminal* subgrammar. The *terminal* subgrammar ag_1 with start symbol I_1^0 contains a rule of the form

$$\left[\begin{array}{l} I_1^0 < o > \rightarrow \bar{x} \\ o = x \end{array}\right.$$

for each basic expression x of the M-grammar. The *start* subgrammar ag_0 with start symbol S contains a rule of the form

$$\left[\begin{array}{l} S < o > \rightarrow I_i^0 < p > \\ o = p \wedge cat(p) \in exportcats(i) \end{array}\right.$$

for the start symbol of each attributed subgrammar. The attribute condition in this rule means that S-trees that are exported by subgrammar i have a syntactic category which is in the set $exportcats(i)$.
For each subgrammar rg_i specified by the M-grammar we can construct an attributed subgrammar ag_i being the 5-tuple $(\bar{N}_i \cup \{S\}, \{\triangleright, \square\} \cup \bar{\Sigma}_i, \bar{P}_i, I_i^0, (T, F_i))$ as follows:

- ag_i has 'domain' (T, F_i), where T is the set of possible S-trees and F_i is a collection of relations of type $T^m \times T, m > 0$. F_i contains all relations defined by the M-rules of subgrammar i.

- The set of production rules of ag_i can be constructed as follows:
  - If rg_i contains a rule of the form $I_i^j \rightarrow \bar{r} I_i^k$, where $\bar{r}$ corresponds with an n-ary meaningful M-rule r, ag_i contains the following attribute grammar rule:

$$\left[\begin{array}{l} I_i^j < o > \rightarrow \bar{r} I_i^k < p_1 > S < p_2 > \ldots \\ \qquad \ldots S < p_n > \triangleright \\ (o,(p_1,\ldots,p_n)) \in \mathcal{R}_r \end{array}\right.$$

    Here, I_i^j and I_i^k are non-terminals of the attributed sugrammar ag_i, S is the start symbol of the complete grammar, the terminal $\bar{r}$ is the name of the M-rule and R_r is the binary relation between S-trees and tuples of S-trees which is defined by M-rule r. The terminal symbol $\triangleright$ marks the end of the scope of the production rule in the strings generated by the grammar. The variables $o, p_1 \ldots p_n$ are the attributes of the rule. All attributes are of type S-tree.
    One possible interpretation of the attribute grammar rule is that the S-tree o is received from non-terminal I_i^j of the current subgrammar. According to the relation defined by M-rule r, the S-tree o corresponds to the S-trees $p_1, \ldots, p_n$. S-tree p_1 is passed to another non-terminal of the current subgrammar, whereas $p_2, \ldots, p_n$ are offered to the start symbol of the attribute grammar.
  - If rg_i contains a rule of the form $I_i^j \rightarrow \bar{r} I_i^k$ where $\bar{r}$ corresponds with unary transformation r, ag_i contains the following attribute grammar rule:

$$\left[\begin{array}{l} I_i^j < o > \rightarrow I_i^k < p > \\ (o,p) \in \mathcal{R}_r \end{array}\right.$$

    Notice that an attribute rule corresponding with a transformation r does not produce the terminal $\bar{r}$.
  - If rg_i contains a rule of the form $I_i^j \rightarrow I_i^k$, the ag_i contains the following attribute grammar rule:

$$\left[\begin{array}{l} I_i^j < o > \rightarrow I_i^k < p > \\ o = p \end{array}\right.$$

  - If rg_i contains a rule of the form $I_i^j \rightarrow \epsilon$ then ag_i contains the following rule:

$$\left[\begin{array}{l} I_i^j < o > \rightarrow \square S < p > \\ o = p \wedge cat(p) \in headcats(i) \end{array}\right.$$

    Rules of this form mark the beginning of a subgrammar. The terminal symbol $\square$ is used for this purpose. The attribute relation is a restriction on the kind of S-trees that is allowed to enter the subgrammar. Only S-trees with a syntactic category in the set $headcats(i)$ are accepted.

The set of all attributed subgrammars can be joined to one single attribute grammar $(N, \Sigma, P, S, (T, F))$ as follows:

- The non-terminal set of the attribute grammar is the union of all non-terminals of all subgrammars, i.e. $N = \bigcup_{i=0}^{M} \bar{N}_i$.
- The terminal set Σ of the attribute grammar is the union of all terminals of all subgrammars (including the terminal subgrammar): $\Sigma = \{\triangleright, \square\} \cup \bigcup_{i=0}^{M} \bar{\Sigma}_i$.
- The set of production rules is the union of all production rules of the subgrammars, $P = \bigcup_{i=0}^{M} \bar{P}_i$.
- The startsymbol of the composed grammar is identical to the the startsymbol S of the start subgrammar. The attribute of the start symbol of an attribute grammar is called the *designated* attribute (Engelfriet (1986)) of the attribute grammar. The *output set* of an attribute grammar is the set of all possible values of its designated attribute.
- The composed grammar has domain (T, F) where $F = \bigcup_{i=0}^{M} F_i$ and T is the set of all possible S-trees.

In the rest of the paper we call an attribute grammar which has been derived from an M-grammar in this way an *attributed* M-*grammar* or amg.

Computational Aspects

Because each meaningful attributed rule r produces the terminal symbol $\bar{r}$ and because each terminal rule x produces terminal symbol $\bar{x}$, the strings of $\mathcal{L}(X)$, the language defined by an amg X, will contain the derivational history of the string itself. The history is partial, because the grammar rules for transformations do not produce a terminal. Moreover, the form of the grammar rules is such that each string is a *prefix* representation of its own derivational history.
Given an amg X, with a set of terminals Σ, a *recognition* function of type $\mathcal{L}(X) \rightarrow 2^T$ can be defined as:

$$\mathrm{MGen}(d) =_{def} \{t \mid S<t> \overset{*}{\underset{X}{\Rightarrow}} d \wedge d \in \Sigma^*\}$$

The reverse of MGen is the *generation* function of type $T \rightarrow 2^{\mathcal{L}(X)}$, which can be defined as:

$$\mathrm{MPars}(t) =_{def} \{d \mid S<t> \overset{*}{\underset{X}{\Rightarrow}} d \wedge d \in \Sigma^*\}$$

These functions can of course be defined for each attribute grammar in this form. However, in the case of amg's the MPars and MGen functions are both *computable* because each M-rule r defines both a computable function and its reverse:

$$\begin{array}{ll} (o, (p_1, ..., p_n)) \in \mathcal{R}_r & \Leftrightarrow \\ o \in f_r(p_1, ..., p_n) & \Leftrightarrow \\ (p_1, ..., p_n) \in f_r^{-1}(o) & \end{array}$$

Because of this property of the M-rules the grammar has two possible interpretations:

- one for recognition purposes with only synthesized attributes, in which the rules can be written as:

$$\left[\begin{array}{l} \bar{I}_i^j <\uparrow o> \rightarrow \bar{r}\bar{I}_i^k <\uparrow p_1> S <\uparrow p_2> \ldots \\ \qquad\qquad \ldots S <\uparrow p_n> \triangleright \\ o \in f_r(p_1, ..., p_n) \end{array}\right.$$

  This interpretation is to be used by MGen in the generation phase of the Rosetta system.

- one for generation purposes with only inherited attributes containing the following type of rules:

$$\left[\begin{array}{l} \bar{I}_i^j <\downarrow o> \rightarrow \bar{r}\bar{I}_i^k <\downarrow p_1> S <\downarrow p_2> \ldots \\ \qquad\qquad \ldots S <\downarrow p_n> \triangleright \\ (p_1, ..., p_n) \in f_r^{-1}(o) \end{array}\right.$$

  The generative interpretation of the rules will be used by MPars in the analysis phase of the Rosetta translation system.

From the definitions of MPars and MGen the *reversibility property* of the grammar follows immediately:

$$d \in \mathrm{MPars}(t) \Leftrightarrow t \in \mathrm{MGen}(d)$$

The reversibility property which has always been one of the tenets of the Rosetta system (Landsbergen (1982)) has recently received the appreciation of other researchers in the field of M.T. as well (Isabelle (1989), Rohrer (1989), van Noord (1990)).
In order to give the M-grammar formalism a place in the list of other linguistic formalisms like LFG, FUG, TG, TAG and GPSG [1], we will investigate some computational aspects of amg's in this section. Given an amg grammar X, we can calculate the value of the designated attribute for an element of $\mathcal{L}(X)$. For this calculation an ordinary context free recognition algorithm (Earley(1970), Leermakers(1991)) can be used. Because the grammar may contain cycles of the form

$$\left[\begin{array}{l} \bar{I}_i^j < o > \rightarrow \bar{I}_i^j < p > \\ (o, p) \in \mathcal{R}_r \end{array}\right.$$

its context-free backbone is not finitely ambiguous. Hence, an amg is not necessarily *off-line parsable* (Pereira and Warren (1983), Haas (1989)). The term *off-line parsable* is somewhat misleading because a two-stage parse process for grammars which are infinitely ambiguous is very well feasible. In the first stage of the parse process, in which the context free backbone is used, a finite representation of the infinitely many parse trees, e.g. in the form of a parse matrix, is determined. Next, in the second stage, the attributes are calculated. However, measure conditions on the attributes are necessary to guarantee termination of the parse process. These measure conditions are constraints on the size (according to a certain measure) of the attribute values that occur in each cycle of the underlying context free grammar.
The generative interpretation of amg X can be used in a straight-forward language generator which generates all corresponding elements of $\mathcal{L}(X)$ for a given value of the designated attribute . Obviously, it can only be guaranteed that the generation process will always terminate if

[1] cf. Perrault (1984) for a comparison of the mathematical properties of these formalisms.

the grammar satisfies some restrictions. Suggestions for grammar constraints in the form of termination conditions for parsing and generation are given in Appelo et al.(1987).

For an insight into the weak generative capacity of the formalism we have to examine the set of yields of the S-trees in the output set of an amg. Let us call this set the *output language* defined by an amg. It is not possible to characterize exactly the set of output languages that can be defined by an amg without defining what the termination conditions are. The precise form of the termination conditions, however, is not imposed by the M-grammar formalism. The formalism merely demands that some measure on the attribute values is defined which garantuees termination of the recognition and generation process. In order to get an idea of the weak generative capacity of the formalism, we assume, for the moment, the weakest condition that guarantees termination. It can be shown that each deterministic Turing Machine can be implemented by means of an amg such that the language defined by the TM is the output language of that amg. Not all grammars that can be constructed in this way satify the termination condition, however. The termination condition is only satisfied by Turing Machines that halt on all inputs, which is exactly the class of machines that define the set of all recursive languages. Consequently, the output languages that can be defined by amg's or M-grammars, in principle, are the languages that can be recognized by deterministic Turing Machines in finite time.

At this point it is appropriate to mention the bifurcation of grammatical formalisms into two classes: the formalisms designed as linguistic *tools* (e.g. PATR-II, FUG, DCG) and those intended to be linguistic *theories* (e.g. LFG, GPSG, GB) (cf. Shieber (1987) for a motivation of this bifurcation). The goals of these formalisms with respect to expressive power are, in general, at odds with each other. While great expressive power is considered to be an advantage of tool-oriented formalisms, it is considered to be an undesirable property of formalisms of the theory type. The M-grammar formalism clearly belongs to the category of linguistic tools.

By strengthening the termination conditions it is possible to restrict the class of output languages that can be defined by an amg. For instance, the class of output languages can be restricted to the languages that are recognizable by a deterministic TM in 2^{cn} time [2] if we assume that the termination conditions imposed on an amg are the weakest conditions that satisfy the constraints formulated in Rounds (1973). A reformulation of these constraints for amg's is as follows:

- The time needed by an attribute evaluating function is proportional to some polynomial in the sum of the size of its arguments.
- There is a positive constant λ such that in each fully attributed derivation tree, the size of each attribute value is less than or equal to the size of the constant λ times the size of the value of the designated attribute.

Rounds used these conditions to show that the languages recognisable in exponential time make up exactly the set which is characterized by transformational grammars (as presented in Chomsky (1965)) satisfying the terminal-length non-decreasing condition.

The power of the formalism with respect to generative capacity has of course its consequences for the computational complexity of the generation and recognition process. Here too, the exact form of the termination condition is important. Obeying the termination conditions that we adhere to in the current Rosetta system, it can be proved that the recognition and the generation problems are NP-hard, which makes them *computationally intractable*. In comparison with other formalisms, M-grammars are no exception with respect to the complexity of these issues. LFG recognition and FUG generation have both been proved to be NP-hard in Barton et al. (1987) and Ritchie (1986) respectively. Recognition in GPSG has even been proved to be EXP-POLY-hard (Barton et al. 1987). We should keep in mind, however, that the computational complexity analysis is a *worst-case* analysis. The average-case behaviour of the parse and generation algorithm that we experience in the daily use of the Rosetta system is certainly not exponential.

Isomorphic Grammars

The decidability of the question whether two M-grammars are isomorphic is another computational aspect related to M-grammars. Although this mathematical issue appears not to be very relevant from a practical point of view, it enables us to show what grammar isomorphy means in the context of amg's.

According to the Rosetta Compositionality Principle (Landsbergen(1987)) to each meaningful M-rule r a meaning rule m_r corresponds which expresses the semantics of r. Furthermore, there is a set of basic meanings for each basic expression of an M-grammar. We can easily express this relation of M-grammar rules and basic expressions with their semantic counterparts in an amg. Instead of incorporating the M-rule name $\bar{r}$ in the attributed production rule as we did in the previous sections, we now include the name of the corresponding meaning rule $\bar{m}_r$ as follows:

$$\left[\begin{array}{l} I_i^j < o > \rightarrow \bar{m}_r \bar{I}_i^k < p_1 > S < p_2 > \ldots S < p_n > \triangleright \\ (o, (p_1, \ldots, p_n)) \in \mathcal{R}_r \end{array}\right.$$

The terminal subgrammar must be adapted in order to generate basic meanings instead of basic expressions. If a basic expression x corresponds with the basic meanings $m_x^1, \ldots, m_x^j, \ldots, m_x^n$ then we replace the original rule in the terminal subgrammar for x by n rules of the form:

$$\left[\begin{array}{l} I_i^0 \rightarrow \bar{m}_x^j \\ o = x \end{array}\right.$$

We will call a grammar that has been derived in this way from an amg a *semantic* amg, or samg. The strings

[2] This includes all context sensitive languages (Cook (1971)).

of the language defined by an samg are prefix representations of semantic derivation trees. The language defined by an samg is called the set of strings which are *well-formed* with respect to X.
Let us repeat here what it means for two M-grammars to be isomorphic:
"...Two grammars are isomorphic iff each semantic derivation tree which is well-formed with respect to one grammar is also well-formed with respect to the other grammar..." (Landsbergen (1987)). We can reformulate the original definition of isomorphic M-grammars in a very elegant way for samg's:

Definition: Two samg's X_1 and X_2 are *isomorphic* iff they are *equivalent*, that is iff $\mathcal{L}(X_1) = \mathcal{L}(X_2)$

This definition says that writing isomorphic grammars comes down to writing two attribute grammars which define the same language. From formal language theory (e.g. Hopcroft and Ullman (1979)) we know that there is no algorithm that can test an arbitrary pair of context-free grammars G_1 and G_2 to determine whether $\mathcal{L}(G_1) = \mathcal{L}(G_2)$. It can also be shown that samg's can define any recursive language. Consequently, checking the equivalence of two arbitrary samg's will be an *undecidable* problem. Rosetta grammars that are used for translation purposes, however, are not arbitrary samg's: they are not created completely independently. The strategy followed in Rosetta to accomplish the definition of equivalent grammars, that is, grammars that define identical languages, is to *attune* two samg's to each other. This *grammar attuning* strategy is extensively described in Appelo et al.(1987), Landsbergen (1982) and Landsbergen (1987) for ordinary M-grammars. Here, we will show what the attuning strategy means in the context of samg's, together with a few extensions.
The attuning measures below must not be looked at as the weakest possible conditions that guarantee isomorphy. The list merely is an enumeration of conditions which together should help to establish isomorphy. If two samg's X_1 and X_2 have to be isomorphic, the following measures are proposed:

- *The production rules of both* samg*'s must be consistent.*
  If both grammars have a production rule in which the name of the meaning rule m appears, then the right-hand side of the rules should contain the same number of non terminals, since m is a function with a fixed number of arguments, independent of the grammar it is used in.
- *The terminal sets of both* samg*'s should be equal*[3].
  In the context of the ordinary M-grammar formalism this condition is formulated as:
  - for each basic expression in one M-grammar there has to be at least one basic expression in the other M-grammar with the same meaning (which comes down to the condition that the terminal set of the terminal subgrammars should be identical)
  - for each meaningful rule in one M-grammar there has to be at least one meaningful rule in the other M-grammar which has the same meaning.
- *The underlying context free grammars of both* samg*'s should be equivalent.*
  Equivalence of the underlying context free grammars can be established by putting an equivalence condition on the underlying grammar of corresponding subgrammars of the samg's in question. Suppose that for each subgrammar of an samg X_1 a subgrammar of another samg X_2 would exist that performs the same linguistic task and vice versa. Such an ideal situation could be expressed by a relation $\mathcal{R}$ on the sets of subgrammars of both samg's. Let i and j be subgrammars of the samg's X_1 and X_2 respectively, such that $(i,j) \in \mathcal{R}$, then the underlying grammars[4] B_i and B_j have to be constructed in such a way that they define the same language. (Notice that B_i and B_j are regular grammars.) More formally:
  $\forall (i,j) \in \mathcal{R} : \mathcal{L}(B_i) = \mathcal{L}(B_j)$. [5]

The three attuning conditions above guarantee that the underlying context free grammars of two attuned samg's are equivalent. However, the language defined by an samg is a subset of the language defined by its underlying grammar. The rule conditions determine which elements are in the subset and which are not. Because of the great expressive power of M-rules, the attuning measures place no effective restrictions on the kind of languages an samg can define. Hence, it can be proved that:

Theorem: The question whether two attuned samg's are *isomorphic* is *undecidable.*

Because of the equivalence between samg's and M-grammars this also applies to arbitrary attuned M-grammars. Future research is needed to find extensions for the attuning measures in a way that guarantees isomorphy if grammar writers adhere to the attuning conditions. The extensions will probably include restrictions on the form of the underlying grammar and on the expressive power of M-rules. Also formal attuning measures between M-rules or sets of M-rules of different grammars are conceivable.

[3] This condition is equivalent to the attuning measures described in Appelo et al. (1987), Landsbergen (1982) and Landsbergen(1987).

[4] Because we are dealing with a subgrammar, the nonterminal S is discarded from the production rules of the underlying grammar.

[5] This attuning measure sketches an ideal situation. In practice for each subgrammar of an samg there is not a corresponding fully isomorphic subgrammar but only a partially isomorphic subgrammar of the other samg. However, the requirement of fully isomorphic subgrammars is not the weakest attuning condition that guarantees the equivalence of the underlying context free grammars. Equivalence can also be guaranteed if X_1 and X_2 satisfy the following condition which expresses partial isomorphy between subgrammars:

$$\bigcup_{i \in X_1} \mathcal{L}(B_i) = \bigcup_{j \in X_2} \mathcal{L}(B_j)$$

The current Rosetta grammars obey the three previously mentioned attuning measures. In practice these measures provide a good basis to work with. Therefore, the undecidability of the isomorphy question is not an urgent topic at the moment.

Conclusions

In this paper we presented the interpretation of an M-grammar as a specification of an attribute grammar. We showed that the resulting attribute grammar is reversible and that it can be used in ordinary context free recognition and generation algorithms. The generation algorithm is to be used in the analysis phase of Rosetta, whereas the recognition algorithm should be used in the generation phase. With respect to the weak generative capacity it has been concluded that the set of languages that can be generated and recognized depends on the termination conditions that are imposed on the grammar. If the weakest termination condition is assumed, the set of languages that can be defined by an M-grammar is equivalent to the set of languages that can be recognized by a deterministic Turing Machine in finite time. Using more realistic termination conditions, the computational complexity of the recognition and generation problem can still be classified as NP-hard and, consequently, as computationally intractable. Finally, it was concluded that the question whether two attuned M-grammars are isomorphic, is undecidable.

Acknowledgements

The author wishes to thank Jan Landsbergen, Jan Odijk, André Schenk and Petra de Wit for their helpful comments on earlier versions of the paper. The author is also indebted to Lisette Appelo for encouraging him to write the paper and to René Leermakers with whom he had many fruitful discussions on the subject.

References

Appelo, L. , C. Fellinger and J. Landsbergen (1987), 'Subgrammars, Rule Classes and Control in the Rosetta Translation System', Philips Research M.S. 14.131, *Proceedings of 3rd ACL Conference* , European Chapter, pp. 118-133.

Barton, G., R. Berwick and E. Ristad (1987), *Computational Complexity and Natural Language*, MIT Press, Cambridge, Mass.

Chomsky, N. (1965), *Aspects of the Theory of Syntax*, MIT Press, Cambridge, Mass.

Cook, S. A. (1971), Characterizations of Pushdown Machines in Terms of Time-bounded Computers, *Journal of the Association for Computing Machinery* 18, 1, pp. 4-18.

Deransart, P., M. Jourdan, B. Lorho (1988), 'Attribute Grammars', *Lecture Notes in Computer Science* 323, Springer-Verlag, Berlin.

Earley, J. (1970), 'An efficient context-free parsing algorithm', *Commun. ACM 13* (1970), pp. 94-102.

Engelfriet, J. (1986), 'The Complexity of Languages Generated by Attribute Grammars', *SIAM Journal on Computing* 15, 1, pp. 70-86.

Haas, A. (1989), 'A Generalization of the Offline Parsable Grammars', *Proceedings of the 27th Annual Meeting of the Association for Computational Linguistics*, pp. 237-242.

Hemerik, C. (1984), 'Formal definitions of programming languages as a basis for compiler construction', Ph.D. th., University of Eindhoven.

Hopcroft, J.E. and J.D. Ullman (1979), 'Introduction to Automata Theory, Languages and Computation', Addison Wesley Publishing Company, Reading, Mass.

Isabelle, P. (1989), 'Towards Reversible M.T. Systems', *MT Summit II*, pp. 67-68.

Knuth, D.E. (1968), 'Semantics of Context-Free Languages', *Math. Systems Theory 2*, 2, pp. 127-145 (June 1968).

Landsbergen, J. (1981), 'Adaptation of Montague grammar to the requirements of parsing', in: *Formal Methods in the Study of Language Part 2*, MC Tract 136, Mathematical Centre, Amsterdam.

Landsbergen, J. (1982), 'Machine Translation based on logically isomorphic Montague grammars', *Coling 82*, North-Holland, Amsterdam, pp. 175-181.

Landsbergen, J. (1987), 'Isomorphic grammars and their use in the Rosetta Translation system', *Machine Translation, the State of the Art*, M. King (ed.), Edinburg University Press.

Leermakers, R (1991), 'Non-deterministic recursive ascent parsing', *Proceedings of the 5th ACL Conference*, European Chapter, forthcoming.

Noord, van G. (1990), 'Reversible Unification Based Machine Translation', in *Proceedings of the 13th International Conference on Computational Linguistics*, Helsinki.

Pereira, F., D. Warren (1983), 'Parsing as deduction', *Proceedings of the 21th Annual Meeting of the Association for Computational Linguistics*, pp. 137-144.

Perrault, C.R. (1984), 'On the Mathematical Properties of Linguistic Theories', *Computational Linguistics* 10, pp. 165-176.

Ritchie, G. (1986), 'The computational complexity of sentence derivation in functional unification grammar', *Proceedings of Coling'86*, pp. 584-586.

Rohrer, C. (1989), 'New directions in MT systems', *MT Summit II*, pp. 120-122.

Rounds, W. (1975), 'A grammatical characterization of the exponential languages', *Proceedings of the 16th Annual Symposium on Switching Theory and Automata*, IEEE Computer Society, New York, pp. 135-143.

Shieber, S. M. (1987), 'Separating Linguistic Analyses from Linguistic Theories', in *Linguistic Theory and Computer Applications*, Academic Press.