ADAPTATION OF MONTAGUE GRAMMAR TO THE REQUIREMENTS OF PARSING

by

Jan Landsbergen

The paper describes a variant of Montague grammar, of which the composition rules have analytical counterparts on which a parsing algorithm can be based. Separate attention is given to the consequences of including rule schemes and syntactic variables in the grammar.

1. INTRODUCTION

In this paper a variant of Montague grammar, called M-grammar, is developed, for which an effective parsing procedure can be designed.

By "Montague grammar" I mean the formalism for defining the syntax and semantics of both natural and formal languages that is described in Montague's paper "Universal Grammar" (MONTAGUE, 1970), henceforth UG. The grammar of the English fragment in "The Proper Treatment of Quantification in Ordinary English" (MONTAGUE, 1973), henceforth PTQ, is a well-known application of this formalism. Montague grammar is especially attractive because of its elegant way of defining an interpretation for a natural language, which it does by means of a syntax-directed translation into a logical language for which an interpretation is defined directly. This makes Montague grammar suitable, in principle, for application in question-answering systems, where the translation from a natural language into a logical language is a useful intermediate step (cf., for instance, BRONNENBERG et al., 1980).

However, Montague grammar is a generative formalism. It generates natural language sentences and their logical forms "in parallel". In a question-answering system a parser is needed, i.e. an effective procedure which assigns to an input question the syntactic derivation tree(s) required for the translation into the logical language. For applications like this the framework of UG has to be adjusted in such a way that for the grammars within the modified framework a parser can be designed.

Obviously such a parser would also be a useful testing tool during the development of a large grammar.

Although the present paper mainly deals with syntactic problems, it gives due attention to the conditions that have to be fulfilled in order to maintain the systematic correspondence between syntax and semantics.

The paper is organized as follows.

In Section 2 M-grammars are defined. The rules of an M-grammar, called M-rules, are compositional rules which construct a labelled bracketing, called S-tree, from "smaller" S-trees and ultimately from basic expressions. In having rules operating on labelled bracketings instead of strings, as in PTQ, I follow the proposal in PARTEE (1973). As Montague's operations on strings can always be reformulated in terms of labelled bracketings, this is not a restriction. The domain in which the M-rules operate, i.e. the set of labelled bracketings that are allowed as possible inputs and outputs of these rules, is defined separately, by means of a context-free grammar. The language defined by the M-grammar is a subset of the language defined by this context-free grammar.

Each compositional M-rule has to obey the condition that it has a unique analytical counterpart. In Section 3 an analytical version of an M-grammar is defined on the basis of these analytical rules. It is proved that the compositional and the analytical version of the grammar define the same language.

Section 4 describes a parsing algorithm based on the analytical version of the grammar. This parser operates as follows. First a context-free parser generates the "surface trees" for the input string, according to the context-free grammar. Then the analytical M-rules are applied to each surface tree.

In Section 5 the framework of M-grammar is extended by the introduction of rule schemes, abbreviations for a possibly infinite set of rules. A modified version of the parser which is able to deal with these rule schemes is defined.

An important feature of Montague grammar is the use of syntactic variables (he_0, he_1, etc.). They are important tools for attaining a systematic relation between syntax and semantics, but they cause problems for a parsing system. These problems and their solution are discussed in Section 6.

In Section 7 a comparison is made with Friedman and Warren's parser for *PTQ and Petrick's parser for Transformational Grammar.*

2. M-GRAMMARS

In this section I will define M-grammars and compare them with Montague grammars as described in UG and PTQ.

An M-grammar is a triple $\langle G_s, B, R\rangle$, where G_s is a loop-free[1] context-free grammar, B is a set of basic expressions, R is a set of M-rules.

The context-free grammar G_s is a quadruple $\langle V_N, V_T, S, P\rangle$, where V_N is a set of syntactic categories, V_T is a set of terminal symbols, S is a distinguished syntactic category and P is a set of production rules. G_S defines a set L_s of labelled ordered trees, called S-trees, in the usual way:

- a node labelled by a terminal symbol is an S-tree.
- if $A \rightarrow B_1 \ldots B_n$ is a production rule and $t_1, \ldots, t_n$ are S-trees with top nodes labelled $B_1, \ldots, B_n$ respectively, then $A[t_1, \ldots, t_n]$ is an S-tree.

I use the labelled bracketing notation $A[t_1, \ldots, t_n]$ for a tree with top node labelled A and $t_1, \ldots, t_n$ as immediate sub-trees. The set of basic expressions B is a subset of L_s. An M-rule $R_i \in R$ has the form of a condition-action pair: $\langle C_i, A_i\rangle$, where C_i is a predicate on n-tuples of S-trees $\langle u_1, \ldots, u_n\rangle$ (if R_i is an n-ary rule). A_i is a function applicable to the n-tuples of S-trees for which C_i holds; $A_i(\langle u_1, \ldots, u_n\rangle)$, i.e. the result of applying function A_i to $\langle u_1, \ldots, u_n\rangle$, is an S-tree. C_i and A_i must be finitely characterizable.

In order to make parsing possible, the M-rules must obey two conditions:

CONDITION C1. For each composition M-rule $R_i = \langle C_i, A_i\rangle$ there is an analytical counterpart $R_i' = \langle C_i', A_i'\rangle$ where C_i' is a predicate applicable to S-trees and A_i' is a function from S-trees to n-tuples of S-trees, such that

(i) C_i' and A_i' can be expressed by means of effective procedures;

(ii) for all $u_1, \ldots, u_n$, $v \in L_s$: $C_i(\langle u_1, \ldots, u_n\rangle)$ and $v = A_i(\langle u_1, \ldots, u_n\rangle)$ if and only if $C_i'(v)$ and $\langle u_1, \ldots, u_n\rangle = A_i'(v)$.

Condition C1 requires that each M-rule has a unique inverse rule. This is a severe condition. In several cases where, from a purely generative point of view, only one rule is needed, we need a rule *scheme* in M-grammars (cf. Section 5).

CONDITION C2. There is a measure for S-trees, i.e. a function μ from S-trees to non-negative integers, such that for each rule r_i holds:
if $C_i(\langle u_1, \ldots, u_n\rangle)$ and $v = A_i(\langle u_1, \ldots, u_n\rangle)$, then for each u_k $(1 \leq k \leq n)$: $\mu(v) > \mu(u_k)$.

An example of a measure is the number of nodes of an S-tree. Condition C2 requires in that case that the results of the application of a compositional rule is an S-tree that is bigger than any of the input S-trees.

In Montague grammar each rule operates on strings of specific syntactic categories and delivers a string of a specific category. In order to maintain this property for M-grammars, we define the syntactic category of an S-tree as the category of its top node and impose Condition C3.

CONDITION C3. For each n-ary rule R_i there are syntactic categories $P_1,\ldots,P_n$, $P_r \in V_N$ such that if $C_i(<u_1,\ldots,u_n>)$ holds, then the categories of the top nodes of $u_1,\ldots,u_n$ are $P_1,\ldots,P_n$, respectively, and the category of the top node of $A_i(<u_1,\ldots,u_n>)$ is P_r.

Because of Condition C3 each rule can be written in a way which is close to the notation in UG:

$$R_i = <<C_i,A_i>,<P_1,\ldots,P_n>,P_r>.$$

The desired systematic relation between syntactic and semantic rules can now be achieved in the usual way: there has to be a mapping ϕ from syntactic categories to semantic types and the semantic composition rule corresponding with R_i must be applicable to expressions of types $\phi(P_1),\ldots,\phi(P_n)$ and build a logical expression of type $\phi(P_r)$.

M-rules that satisfy Condition C3, still differ from the rules in UG in being *partial* rules.[2] The actions A_i only have to be applicable to S-trees that satisfy Condition C_i.

The M-rules are the actual rules of the grammar. They are composition rules, which build an S-tree, starting from the basic expressions. The context-free grammar G_s defines the domain in which the M-rules operate.

The way in which the M-rules construct an S-tree can be indicated by means of a derivation tree, or D-tree. A D-tree is a labelled ordered tree with basic expressions at the terminal nodes and indices i of M-rules R_i at the non-terminal nodes. I will use the notation $i<d_i,\ldots,d_n>$ for a derivation tree with index i at the top node and immediate sub-trees $d_1,\ldots,d_n$.

With each S-tree v that can be generated by the M-rules a D-tree corresponds which shows the derivational history of v. There may be more than one D-tree for v; in that case v is ambiguous with respect to the M-rules.

It should be noted that the applicability of an M-rule depends only

on the input S-trees, not on their D-trees.

The set of D-trees of an arbitrary S-tree v is defined as follows:

$$\text{D-trees}(v) =_{\text{def.}} \text{if } v \in B \text{ then } \{v\}$$
$$\text{else } \{i\langle d_1,\ldots,d_n\rangle \mid \exists u_1,\ldots,u_n \in L_s,\ R_i \in R:$$
$$C_i(\langle u_1,\ldots,u_n\rangle) \wedge v = A_i(\langle u_1,\ldots,u_n\rangle)$$
$$\wedge \forall j:\ 1 \leq j \leq n \Rightarrow d_j \in \text{D-trees}(u_j)\}$$

Less formally: if v is a basic expression, v is its own derivation tree. The derivation trees of an S-tree v that is not a basic expression have a top node labelled with an index i of an M-rule and $d_1,\ldots,d_n$ as immediate sub-trees. In that case there must be a tuple of S-trees $\langle u_1,\ldots,u_n\rangle$ such that $d_1,\ldots,d_n$ are their D-trees and such that M-rule R_i is applicable, delivering the S-tree v.

Let $\sigma(v)$ be the terminal string of v: the sequence of terminal symbols at the terminal nodes of the S-tree v. The language L defined by an M-grammar is the set of terminal strings of S-trees with top node S that have a derivation tree.

$$L =_{\text{def.}} \{\phi \mid \exists v \in L_s:\ \phi = \sigma(v) \wedge \text{D-trees}(v) \neq \emptyset \wedge$$
$$\wedge\ \text{top-node-cat}(v) = S\}.$$

Example of a simple M-grammar

$$G_M = \langle G_s, B, R\rangle.$$

The contex-free grammar G_s is defined as follows:

V_N = {t,T,IV}
V_T = {John, Mary, and, walk, talk}
The distinguished symbol is t.
The production rules are: t → T + IV

$$T \rightarrow T + \text{and} + T$$
$$T \rightarrow \begin{Bmatrix}\text{John}\\ \text{Mary}\end{Bmatrix}$$
$$IV \rightarrow \begin{Bmatrix}\text{walk}\\ \text{talk}\end{Bmatrix}$$

The basic expressions of the M-grammar are:

B = {T[John], T[Mary], IV[walk], IV[talk]}.

The set of M-rules R consists of two rules: R1 and R2. In this paper no specific notation for the M-rules is prescribed. I will explain the notation used in the examples in an ad hoc manner.

RULE R1. $C_1(\langle u_1,u_2\rangle) =_{def.} u_1 = T[T[\ ],\ and,\ T[\ ]] \wedge u_2 = IV[\ ]$.

Here IV[ ] is the notation for an arbitrary S-tree with category IV at the top. So Rule R1 is applicable to a pair $\langle u_1,u_2\rangle$ if u_1 is of category IV.

$A_1(\langle u_1,u_2\rangle) =_{def.} t[u_1,u_2]$.

So the result of A_1 is an S-tree with category t at the top and u_1, u_2 as immediate sub-trees.

RULE R2. $C_2(\langle u_1,u_2\rangle) =_{def.} u_1 = T[\ ] \wedge u_2 = T[\ ]$

$A_2(\langle u_1,u_2\rangle) =_{def.} T[u_1,\ and,\ u_2]$.

The context-free grammar generates S-trees like t[T[John], IV[walk]] (fig. 1), t[T[T[John], and, T[Mary]], IV[walk]] (fig. The S-tree of fig. 1 cannot be generated by the M-rules. The S-tree of fig can be generated by the M-rules and the corresponding derivation tree is 1<2<T[John], T[Mary]>, IV[walk]> (fig. 3).

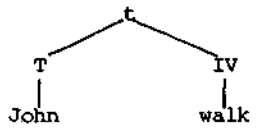

fig. 1

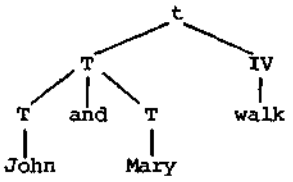

fig. 2

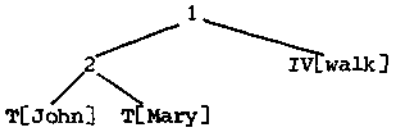

fig. 3

It can easily be checked that Rules R1 and R2 obey Conditions C1, C2 and C3, if we choose the number of nodes as the measure function.

The analytical versions of Rules R1 and R2 are:

RULE R1'. $C_1'(v) =_{def.} v = t[T[T[\ ], \text{and}, T[\ ]], IV[\ ]]$.

$A_1'(v) =_{def.} \langle v.1, v.2\rangle$.

Here $v_j.i$ is the notation for the i-th immediate sub-tree of v_j.

RULE R2'. $C_2'(v) =_{def.} v = T[T[\ ], \text{and}, T[\ ]]$

$A_2'(V) =_{def.} \langle v.1, v.3\rangle$.

3. THE ANALYTICAL VERSION OF AN M-GRAMMAR AND ITS EQUIVALENCE TO THE COMPOSITIONAL VERSION

Because the rules of an M-grammar have to satisfy Condition C1, there is an analytical Rule $R_i' = \langle C_i', A_i'\rangle$ corresponding with each composition rule $R_i = \langle C_i, A_i\rangle$. If R' is a set of such analytical rules we will call $\langle G_s, B, R'\rangle$ an analytical M-grammar. An M-grammar as defined in Section 2 will be called a compositional grammar. For each compositional grammar there is an analytical version.

The set of derivation trees that an analytical grammar assigns to an S-tree is defined as follows:

$$\text{D-trees}_{an}(v) =_{def.} \text{if } v \in B \text{ then } \{v\}$$
$$\text{else } \{i\langle d_1,\ldots,d_n\rangle \mid \exists u_1,\ldots,u_n \in L_s,\ R_i' \in R':$$
$$C_i'(v) \wedge \langle u_1,\ldots,u_n\rangle \approx A_i'(v) \wedge$$
$$\forall j:\ 1 \le j \le n \Rightarrow d_j \in \text{D-trees}_{an}(u_j)\}.$$

The definition suggests how the derivation trees of v can be found by top-down application of the analytical rules. In Section 4 a parser will be described based upon this definition. In this section I prove the equivalence of the analytical and the compositional version of an M-grammar.

I will call the two versions equivalent if they assign to each string the same set of derivation trees. As both versions use the same context-free grammar G_s, it is sufficient to prove that they assign to any S-tree v the same set of derivation trees.

LEMMA 1. $\forall v \in L_s$: $d \in$ D-trees(v) $\Rightarrow d \in \text{D-trees}_{an}(v)$.

PROOF. We use induction on the number of nodes in d: #nodes(d).

The analytical versions of Rules R1 and R2 are:

RULE R1'. $C_1'(v) =_{def.}$ v = t[T[T[ ], and, T[ ]], IV[ ]].

$A_1'(v) =_{def.}$ <v.1, v.2>.

Here v_j.i is the notation for the i-th immediate sub-tree of v_j.

RULE R2'. $C_2'(v) =_{def.}$ v = T[T[ ], and, T[ ]]

$A_2'(V) =_{def.}$ <v.1, v.3>.

3. THE ANALYTICAL VERSION OF AN M-GRAMMAR AND ITS EQUIVALENCE TO THE COMPOSITIONAL VERSION

Because the rules of an M-grammar have to satisfy Condition C1, there is an analytical Rule $R_i' = \langle C_i', A_i' \rangle$ corresponding with each composition rule $R_i = \langle C_i, A_i \rangle$. If R' is a set of such analytical rules we will call $\langle G_s, B, R' \rangle$ an analytical M-grammar. An M-grammar as defined in Section 2 will be called a compositional grammar. For each compositional grammar there is an analytical version.

The set of derivation trees that an analytical grammar assigns to an S-tree is defined as follows:

$$\text{D-trees}_{an}(v) =_{def.} \text{if } v \in B \text{ then } \{v\}$$
$$\text{else } \{i\langle d_1,\ldots,d_n\rangle \mid \exists u_1,\ldots,u_n \in L_s,\ R_i' \in R':$$
$$C_i'(v) \wedge \langle u_1,\ldots,u_n\rangle = A_i'(v) \wedge$$
$$\forall j:\ 1 \le j \le n \Rightarrow d_j \in \text{D-trees}_{an}(u_j)\}.$$

The definition suggests how the derivation trees of v can be found by top-down application of the analytical rules. In Section 4 a parser will be described based upon this definition. In this section I prove the equivalence of the analytical and the compositional version of an M-grammar.

I will call the two versions equivalent if they assign to each string the same set of derivation trees. As both versions use the same context-free grammar G_s, it is sufficient to prove that they assign to any S-tree v the same set of derivation trees.

LEMMA 1. $\forall v \in L_s$: $d \in \text{D-trees}(v) \Rightarrow d \in \text{D-trees}_{an}(v)$.

PROOF. We use induction on the number of nodes in d: #nodes(d).

1. #nodes(d) = 1.

If a derivation tree d in D-trees(v) consists of one node, v must be a basic expression. In that case

$$\text{D-trees}(v) = \text{D-trees}_{an}(v) = v.$$

So the theorem holds for all d with #nodes(d) = 1.

2. Assume that the theorem holds for all d with

$$\#\text{nodes}(d) \leq k.$$

3. Proof for #nodes(d) = k + 1.

d has more than one node and therefore must have the form $i\langle d_1,\ldots,d_n\rangle$. According to the definition of D-trees(v):

(1) $\exists u_1,\ldots,u_n \in L_s,\ R_i \in R$ such that $C_i(\langle u_1,\ldots,u_n\rangle)$,
$v = A_i(\langle u_1,\ldots,u_n\rangle)$ and $d_j \in \text{D-trees}(u_j) \quad (1 \leq j \leq n)$.

From $C_i(\langle u_1,\ldots,u_n\rangle)$ and $v = A_i(\langle u_1,\ldots,u_n\rangle)$ it follows (Condition C1) that $\exists R_i' \in R'$ such that $C_i'(v)$ holds and $\langle u_1,\ldots,u_n\rangle = A_i'(v)$. Because #nodes(d) = k + 1, #nodes(d_j) ≤ k. According to the induction hypothesis: $d_j \in \text{D-trees}(u_j) \Rightarrow d_j \in \text{D-trees}_{an}(u_j)$. So from (1) we can deduce (2).

(2) $\exists u_1,\ldots,u_n \in L_s,\ R_i' \in R'$ such that $C_i'(v)$,
$\langle u_1,\ldots,u_n\rangle = A_i'(v) \wedge d_j \in \text{D-trees}_{an}(u_j), \quad (1 \leq j \leq n)$. □

From (2) it follows immediately that $d \in \text{D-trees}_{an}(v)$.

LEMMA 2. $\forall v \in L_s$: $d \in \text{D-trees}_{an}(v) \Rightarrow d \in \text{D-trees}(v)$.

PROOF. Completely analogous to the proof of Lemma 1. □

THEOREM. $\forall v \in L_s$: $\text{D-trees}(v) = \text{D-trees}_{an}(v)$.

PROOF. The theorem follows immediately from Lemma 1 and Lemma 2. □

4. A PARSER FOR M-GRAMMARS

On the basis of the definition of $\text{D-trees}_{an}(v)$, a procedure M-PARSER can be designed which assigns to an S-tree v its set of derivation trees. I present here the main structure of this procedure. It is so close to the definition of D-trees_{an} that one can trust that it indeed delivers the set of D-trees defined by the analytical M-grammar. The proof of Section 3

guarantees that this is the set of D-trees defined by the compositional M-grammar.

```
M-PARSER(v):
begin
S_D := ∅;
if v ∈ B
then S_D := {v}
else for each analytical rule R'_i ∈ R' do
     if C'_i(v)
     then begin
        <u_1,...,u_n> := A'_i(v);
        for each tuple <d_1,...,d_n> ∈
        M-PARSER(u_1) × ... × M-PARSER(u_n) do
        S_D := S_D ∪ {i<d_1,...,d_n>}
        end;
M-PARSER := S_D
end
```

M-PARSER applies the analytical M-rules to the S-tree v in a top-to-bottom fashion. S_D is the set of D-trees, originally empty. Successful application of a rule R'_i to v results in a tuple $\langle u_1,\ldots,u_n\rangle$. M-PARSER is then applied to $u_1,\ldots,u_n$. Each application of M-PARSER to a u_j gives a (possibly empty) set of D-trees for u_j. For each tuple of D-trees $\langle d_1,\ldots,d_n\rangle$ in the cartesian product of these sets, a D-tree $i\langle d_1,\ldots,d_n\rangle$ is constructed and added to S_D. M-PARSER comes to a successful end if it is ultimately, at the deepest level of recursion, applied to basic expressions.

The procedure M-PARSER is effective, i.e. it ends after a finite number of steps. This follows immediately from the fact that C'_i and A'_i are effective procedures and that each application of $A'_i(v)$ results in a tuple of S-trees with a measure smaller than $\mu(v)$. Because of this the maximal recursion depth of M-PARSER cannot be more than the measure of the S-tree to which it is applied.

I assume here that the number of rules R'_i is finite. In Section 5 I will discuss rule schemes, which may define an infinite number of rules.

M-PARSER is not yet a complete parser. It has to be preceded by an ordinary context-free parser, called CF-PARSER, which assigns to a string s all S-trees that have s as their terminal string and a top node labelled S. Thanks to the requirement that the context-free grammar is loop-free,

there is always such a parser. The complete algorithm is:

```
PARSER(s):
   begin S_D := ∅;
         for each v ∈ CF-PARSER(s) do
         S_D := S_D ∪ M-PARSER(v);
         PARSER := S_D
   end
```

5. RULE SCHEMES

The parsing procedure of Section 4 is only effective if the number of M-rules is finite. In this section I will define rule schemes, abbreviations of a (possibly) infinite set of rules, and describe a parser for an M-grammar with such rule schemes. In Montague grammars rule schemes occur: "rule" S3 of PTQ, for instance, is in fact a rule scheme, with an instance for each variable index. In M-grammars rule schemes are needed more often than in purely generative grammars, because of the condition that for each single rule there is a unique inverse rule.

A rule scheme is a triple S = <P,I,A>.

- P is a possibly infinite set of parameter values.
- I is a function from n-tuples of S-trees $\langle u_1,\ldots,u_n\rangle$ to subsets of P, for an n-ary rule scheme.
- A is a function with two arguments: a parameter value p and an n-tuple of S-trees $\langle u_1,\ldots,u_n\rangle$; the result of the application of A is an S-tree.

P, I and A must be finitely characterizable.

In an ordinary M-rule <C,A>, condition C decides whether or not the rule is applicable to a particular tuple $\langle u_1,\ldots,u_n\rangle$ of S-trees. A rule scheme <P,I,A> may be applicable in several ways and each parameter value in $I(\langle u_1,\ldots,u_n\rangle)$ determines one way in which the rule scheme is applicable.

Each rule scheme <P,I,A> defines a set of M-rules $\{\langle C_p,A_p\rangle \mid p \in P\}$, where

$$C_p(\langle u_1,\ldots,u_n\rangle) =_{def.} p \in I(\langle u_1,\ldots,u_n\rangle)$$

$$A_p(\langle u_1,\ldots,u_n\rangle) =_{def.} A(p,\langle u_1,\ldots,u_n\rangle).$$

A rule $\langle C_p,A_p\rangle$ is called an instance of the rule scheme.

Rule schemes have to obey three conditions: CS1, CS2 and CS3.

CONDITION CS1. For each rule scheme $S = \langle P,I,A\rangle$ there is an analytical rule scheme $S' = \langle P,I',A'\rangle$, such that

- I' is a function from S-trees to *finite* subsets of P.
- A' is a function with two arguments: a parameter $p \in P$ and an S-tree; the value of A' is an n-tuple of S-trees.
- I' and A' can be expressed by means of effective procedures.
- $\forall u_1,\ldots,u_n,\ v \in L_s$:
  $p \in I(\langle u_1,\ldots,u_n\rangle) \wedge v = A(p,\langle u_1,\ldots,u_n\rangle)$ if and only if
  $p \in I'(v) \wedge \langle u_1,\ldots,u_n\rangle = A'(p,v)$.

CONDITION CS2. Condition C2 must hold for all instances of the rule scheme.

CONDITION CS3. Condition C3 must hold for all instances of the rule scheme.

An analytical rule scheme $\langle P,I',A'\rangle$ defines a set of analytical M-rules $\{\langle C'_p,A'_p\rangle \mid p \in P\}$, where

$$C'_p(v) =_{def.} p \in I'(v)$$
$$A'_p(v) =_{def.} A'(p,v).$$

From these definitions and Condition CS1 it follows immediately that each instance $\langle C_p,A_p\rangle$ obeys Condition C1 and that $\langle C'_p,A'_p\rangle$ is its analytical counterpart.

The conclusion can be that from an M-grammar with rule schemes a compositional and an analytical M-grammar, as defined in Sections 2 and 3, can be derived. The functions D-trees and $D\text{-trees}_{an}$ can be expressed in terms of these derived M-grammars. The equivalence proof of Section 3 is valid for M-grammars with an infinite set of rules and therefore is also valid for grammars with rule schemes.

The procedure M-PARSER of Section 4 has to be adjusted. For a grammar with a finite number of rule schemes S'_i it becomes:

```
M-PARSER'(v):
begin
S_D := ∅;
if v ∈ B
then S_D := {v}
else for each rule scheme S'_i do
```

for each parameter value $p \in I_i'(v)$ do
begin
$\langle u_1,\ldots,u_n\rangle := A_i'(p,v)$;
for each tuple $\langle d_1,\ldots,d_n\rangle \in$ M-PARSER'$(u_1) \times \ldots \times$ M-PARSER'(u_n) do
$S_D := S_D \cup \{i_p\ \langle d_1,\ldots,d_n\rangle\}$
end;
M-PARSER' := S_D
end

The effectiveness of M-PARSER' can be established in a similar way as was done for M-PARSER, taking into account that there is a finite number of rule schemes S_i and that $I_i'(v)$ is a finite set.

The correctness of M-PARSER' can be established in the same way as the correctness of M-PARSER, if we realize that all parameter values $p \in I_i'(v)$ specify exactly the applicable instances of the rule schemes S_i'.

The parser has been described in terms of rule schemes only. This is no restriction, because an M-rule can always be considered as a special case of a rule scheme, where the set P of parameter values has only one element.

Example

I will now describe an M-rule scheme for PTQ rule S4. The original rule is:

RULE S4. If u_1 is an expression of category T and u_2 is an expression of category IV, then $u_1 u_2'$ is an expression of category t, where u_2' is the result of replacing the first basic verb in u_2 by its third person singular present.

This rule cannot be represented by a single M-rule, because it does not have a unique inverse rule. The point is that the first basic verb in the IV might, in principle, be preceded by a verb that is already in third person singular present form.

The M-rule scheme for S4 is the triple $\langle P_4, I_4, A_4\rangle$, where P_4 is the set of all possible paths in S-trees. A path is a sequence of branches from the top to a sub-tree. As each branch can be represented by a positive integer (n represents the branch to the n-th daughter), a path can be represented by a sequence of positive integers $i_1 \ldots i_n$. So P_4 is the set of all such sequences. We define:

$i_1 \ldots i_m < j_1 \ldots j_n$ if $\exists k$ $(1 \leq k \leq n)$: $i_1 = j_1, \ldots, i_k = j_k$, $i_{k+1} < j_{k+1}$.

$I_4(\langle u_1, u_2 \rangle) =_{def.}$ {p | u_1 = T[ ] ∧ u_2 = IV[ ] ∧ u_2.p is the first basic verb in u_2}.

For a given pair $\langle u_1, u_2 \rangle$ there is at most one such path p, which illustrates that from a purely generative point of view a rule scheme is not needed.

$A_4(\langle u_1, u_2 \rangle) =_{def.}$ $t[u_1, u_2']$ where u_2' is the result of replacing in u_2 the basic verb u_2.p by its third person singular present.

The inverse rule scheme is $\langle P_4, I_4', A_4' \rangle$.

$I_4'(v) =_{def.}$ {p | v = t[T[ ], IV[ ]] ∧ v.2.p is a verb in third person singular present form ∧ $\neg\exists p_i < p$ such that v.2.p_i is a basic verb}.

The definition of I_4' shows that if, for instance, v.2 contains exactly two verbs, both in third person singular present form, two instances of the inverse rule scheme are applicable.

$A_4'(v) =_{def.}$ $\langle$v.1, $v_2'\rangle$, where v_2' is the result of replacing in v.2 the verb v.2.p by its basic form.

6. SYNTACTIC VARIABLES

An important feature of Montague grammar is the use of syntactic variables. The PTQ grammar for instance, contains an infinite number of variables he_0, he_1, he_2, etc. as basic expressions.[3] It also contains rules that eliminate a variable by substituting a term for it. PTQ rule S3 is such a 'variable-removing' rule; more precisely, it is a rule scheme, with instances for each variable. The problem with rule schemes like S3 is that they assign an infinite number of derivation trees to each sentence, most of them only differing in the choice of their variables.

If we assume that there is a systematic correspondence between

syntactic variables and variables of the logic, as expressed by Janssen's Variable Principle (JANSSEN, 1980), we are able to partition the infinite set of derivation trees of a sentence into a finite number of equivalence classes. The members of an equivalence class of derivation trees differ only in the choice of their variables and, therefore, correspond with logical expressions that are logically equivalent. From this point of view only one 'canonical' derivation tree of each equivalence class is of interest. I will define this canonical derivation tree as follows.

First the *level* of a node in a D-tree is defined as the length of the path (the number of branches) from the top node to that node.

A *variable-removing node* in a D-tree is a node corresponding with a rule that eliminates some variable he_k. According to the Variable Principle the corresponding translation rule into the logic binds the corresponding logical variable.

A *canonical D-tree* is a D-tree where a variable-removing node at level N removes the variable he_N.[4]

Figure 4 shows one of the D-trees assigned by the rules of PTQ to the sentence 'John loves every woman'. Figure 5 shows the corresponding canonical D-tree. At the nodes of these D-trees the index of the corresponding PTQ rules and the removed variables are indicated (index 2^a stands for the first sub-rule of S2, where syntactic operation F_0 places 'every' before a CN).

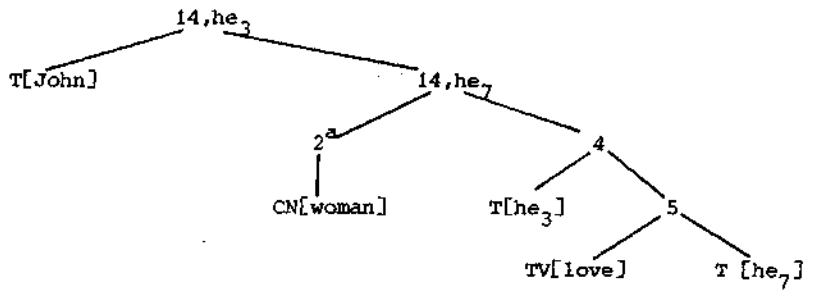

Figure 4

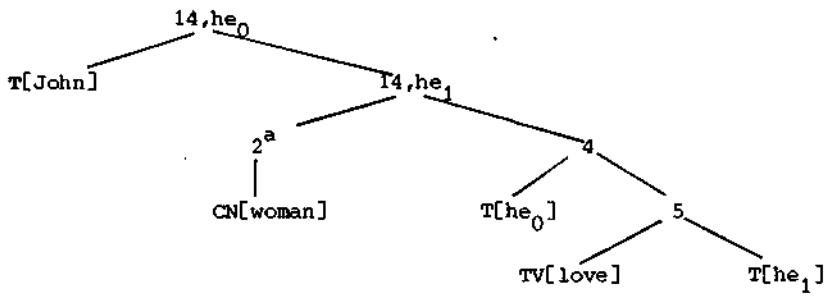

Figure 5

I will assume here, without proof, that each of the above-mentioned equivalence classes of D-trees contains one canonical D-tree. In that case a parser only needs to generate canonical D-trees. The assumption is only correct if each variable-removing rule is an instance of a rule scheme with instances for each variable. Let <P,I',A'> be the analytical version of such a rule scheme (each instance is a 'variable-introducing' rule).

We will replace I' by a function I", which has a second argument, N. Now, I"(v,N) does not give all parameter values for v, but only those that are relevant for the rule instances that introduce variable he_N. For analytical rule schemes that do not introduce a variable, I"(v,N) is equal to the original I'(v).

If the parser tries to construct a new node of the derivation tree at level N_1 by applying rule scheme <P,I',A'> it has to call I"(v,N_1). This can be achieved by giving the new parsing procedure, M-PARSER", the level N as a second argument. If the parsing process is started by calling M-PARSER"(v,0), only canonical derivation trees are generated.

```
M-PARSER"(v,N):
begin
S_D := ∅;
if v ∈ B
then S_D := {v}
else for each rule scheme S'_i do
     for each parameter value p ∈ I"_i(v,N) do
     begin
     <u_1,...,u_n> := A'_i(p,v);
     for each tuple <d_1,...,d_n> ∈ M-PARSER"(u_1,N+1) × ... × M-PARSER"(u_n,N+1)
```

```
        do S_D := S_D ∪ {i_p <d_1,...,d_n>}
        end;
    M-PARSER" := S_D
    end
```

The complete parser becomes:

```
    PARSER"(s):
    begin S_D := ∅;
          for each v ∈ CF-PARSER(s) do
              S_D := S_D ∪ M-PARSER"(v,0);
          PARSER" := S_D
    end
```

The effectivity of M-PARSER" follows immediately from the effectivity of M-PARSER', and the fact that $I_i'(v,N)$ delivers a finite set of parameter values.

However, M-PARSER' and M-PARSER" are only effective if a measure can be defined such that each rule satisfies Condition C2. The number of nodes is not an adequate measure here, because rules that substitute a single term ('John' for example) for a variable, like PTQ rule S14(i), would not increase the number of nodes. Therefore, we choose the number of 'non-variable' nodes as the measure. This still precludes rules like PTQ rule (scheme) S14(ii), which only substitutes a variable for another variable. These rules are superfluous from a semantic point of view. They seem to have been included by Montague in order to make S14 a total rule. This measure also precludes the 'vacuous' applications of S14 that arise when a term is substituted for a variable that is not present in the expression. FRIEDMAN & WARREN (1978) already noticed that these applications of the rule lead to incorrect translations into the logic.

Example

As an example of a rule scheme involving syntactic variables I will give the M-grammar formulation of PTQ rule scheme S3. The original PTQ rule is:

RULE S3. If u_1 is an expression of category CN and u_2 is an expression of category t, then "u_1 such that u_2'" is an expression of category CN, where u_2' comes from u_2 by replacing each occurrence of he_n or

him_n by he, she, it or him, her, it, respectively, according as the first basic expression of category CN in u_1 is of masc., fem. or neuter gender.

The M-rule scheme for S3 is $\langle P_3, I_3, A_3 \rangle$, where

$$P_3 = \{\langle g,Q,n\rangle \mid g \in \{\text{masc., fem., neuter}\},\ Q \text{ is a set of paths},\ n \text{ is a variable index}\}.$$

Each parameter value is a triple consisting of a gender, a set of paths and a variable index. (It is not absolutely necessary to include the gender in the parameter, but it facilitates the formulation of the rule scheme.)

$$I_3(\langle u_1,u_2\rangle) =_{def.} \{\langle g,Q,n\rangle \mid n \geq 0,\ g \text{ is the gender of the first terminal CN in } u_1,\ Q = \{p \mid u_2.p = he_n \vee u_2.p = him_n\}\}.$$

So for a given pair $\langle u_1,u_2\rangle$ there is a parameter value $\langle g,Q,n\rangle$ for each variable index n, where Q is the set of all paths from the top of u_2 to variables with this index. Obviously, $I_3(\langle u_1,u_2\rangle)$ is an infinite set, which contains an infinite number of triples of the form $\langle g,\emptyset,n\rangle$, where n is the index of a variable not occurring in u_2, and $\emptyset$ is the empty set.

$A_3(\langle g,Q,n\rangle, \langle u_1,u_2\rangle) =_{def.}$ CN[u_1, such that, u_2'], where u_2' is the result of replacing in u_2 for each $p_i \in Q$ the variable at $u_2.p_i$ by the pronoun of the appropriate case and gender.

The inverse rule scheme is $\langle P_3, I_3', A_3' \rangle$, where

$I_3'(v) =_{def.}$ $\{\langle g,Q,n\rangle \mid$ v = CN[CN[ ], such that, t[ ]] $\wedge$ g is the gender of first terminal CN in v.1 $\wedge$ v.3 does not contain variables with index n $\wedge$ Q is a subset of the set of paths to pronouns with gender g in v.3$\}$.

$A_3'(\langle g,Q,n\rangle,v) =_{def.}$ $\langle u_1,u_2\rangle$, where u_1 = v.1 and u_2 is the result of replacing in v.2 for all $p_i \in Q$ the pronoun v.2.p_i by the variable he_n or him_n according to the case of the pronoun.

$I'_3(v)$ is an infinite set. The subset of parameter values relevant for the rule instances that introduce variable he_N or him_N is defined as:

$$I'_3(v,N) =_{def.} \{\langle g,Q,n\rangle \mid \langle g,Q,n\rangle \in I'_3(v) \wedge n=N\}.$$

7. CONCLUDING REMARKS

The notion 'M-grammar' as described in this paper meets the standard I formulated in the introduction: it is a variant of Montague grammar for which effective parsers can be designed, and which maintains the systematic relation between the syntactic rules and the translation rules into the logic.

As an exercise in the new formalism, the complete grammar of PTQ has been rewritten in the form of an M-grammar and the corresponding parser has been programmed.

It may be interesting to compare the parser described here with Friedman and Warren's parsing method for Montague grammars (FRIEDMAN & WARREN, 1978), though this comparison is made difficult because of an important difference between their approach and mine. F. & W.'s parser is more or less tuned to PTQ, while my goal was primarily to define a class of grammars for which the same kind of parsing algorithm can be used. F. & W. describe the implementation of the individual PTQ rules in detail, whereas I confine myself to the global algorithmic structure of the program. A major problem with F. & W.'s parser - based upon an Augmented Transition Network representation - is that it is difficult to establish its correctness. Especially after reading the passage in F. & W.'s paper (pp.366-368) about the way in which false parsers are avoided in the case of terms nested in other terms, one is left with the uneasy feeling that other complicated cases might have been overlooked.

In the case of M-grammars the parser can be derived systematically from the analytical version of the grammar, which can be proved to be equivalent to the original compositional version. The equivalence of the compositional and the analytical version of each individual rule must be checked, but one does not have to bother about the interaction with other rules. Therefore, it is possible to have confidence in the correctness of the parser. The price paid for this is that there is some redundancy in the grammar and accordingly in the parser. If there are M-rules that are in fact context-free (as is the case for several PTQ rules), they are

duplicated in the context-free grammar.

Another comparison that can be made is with Petrick's recognition procedure for Transformational Grammar (PETRICK, 1965). In order to make effective recognition and parsing possible, Petrick imposes two conditions on TG:

1. a recoverability condition on transformations, comparable with but weaker than my Condition C1;
2. a condition on the depth of S-embedding in the deep structure.

The second condition has the same objective as my condition C2: to guarantee that the parsing procedure comes to an end after a finite number of steps. However, Petrick's second condition is not a restriction on the individual rules, but on the whole collection of rules.

Obviously, requiring that transformations make the syntactic trees bigger, according to some measure, would be unacceptable in TG. Transformations usually have a kind of paraphrasing character and may even involve deletions. In Montague grammar, the rules are basically compositional, they build a new expression from parts that are intuitively smaller. Condition C2 requires expressions that are intuitively smaller to be smaller in a technical sense as well. Though I follow Partee's suggestion to apply Montague rules to labelled trees instead of strings, I do not support her proposal to incorporate transformation rules in Montague grammar. It seems worth while to investigate first what is possible within the more restrictive framework of compositional rules.

ACKNOWLEDGEMENTS. I wish to thank W.J.H.J. Bronnenberg, H.C. Bunt, J.H.G. Rous, R.J.H. Scha and editor T.M.V. Janssen for their comments on an earlier version of this paper. J.H.G. Rous programmed the PTQ parser.

FOOTNOTES

1. A context-free grammar is called loop-free if derivations of the form $A \overset{*}{\Rightarrow} A$ are not possible.
2. If the rules are total, i.e. applicable to all expressions of the required syntactic categories, the translation into the logic can be defined in an elegant way, i.e. as a homomorphism between algebras. This makes it possible to prove that an interpretation of the logic (again a homomorphism into an algebra) induces an interpretation of the natural

language. Therefore, it may be interesting to note that total rules can always be derived from the partial rules of M-grammar in a trivial way, by redefining the set of syntactic categories:

- Introduce a category $P(u_k)$ for each S-tree u_k. S-tree u_k is the one and only S-tree of category $P(u_k)$.
- Replace each rule R_i by an infinite number of new rules, as follows: if $C_i(\langle u_1,\ldots,u_n\rangle)$ holds, then
  $\langle A_{i,P(u_1),\ldots,P(u_n)}, \langle P(u_1),\ldots,P(u_n)\rangle, P(u_r)\rangle$
  is a new, total, rule, where
  $A_{i,P(u_1),\ldots,P(u_n)}(\langle u_1,\ldots,u_n\rangle) =_{def.} A_i(\langle u_1,\ldots,u_n\rangle)$.

On the basis of these rules a many-sorted algebra of derivation trees can be defined, with a sort for each category $P(u_i)$ and operations $A_{i,P(u_1),\ldots,P(u_n)}$. The translation into the logical language can then be defined as a homomorphism from this algebra into an algebra of logical expressions.

If total rules are preferred for other reasons (cf. JANSSEN, 1978), M-grammar allows them, of course, as special cases of partial rules.

3. The introduction of an infinite set of variables is not in conflict with the condition that the set of possible S-trees must be defined by means of a context-free grammar, with a finite set of terminals. The variables are basic expressions, but not necessarily terminals. They may be compound S-trees, defined with the help of context-free rules.
4. I assume here, for the sake of simplicity, that all variables have the same syntactic category (as in PTQ).

REFERENCES

BRONNENBERG, W.J.H.C., BUNT, H.C., LANDSBERGEN, S.P.J., SCHA, R.J.H., SCHOENMAKERS, W.J. & E.P.C. VAN UTTEREN, 1980, 'The question-answering system PHLIQA 1', in: L. Bolc (ed.), *Natural Language Question-Answering Systems*, Carl Hanser Verlag, München & Wien, 1980, pp.217-305.

FRIEDMAN, J. & D.S. WARREN, 1978, 'A parsing method for Montague grammars', *Linguistics and Philosophy*, 2.

JANSSEN, T.M.V., 1980, 'On problems concerning the quantification rules in Montague grammar', in: G. Rohrer (ed.), *Time, Tense and Quantifiers*, Max Niemeyer Verlag, Tübingen, 1980, pp. 113-134.

JANSSEN, T.M.V., 1978, 'Compositionality and the form of the rules in Montague grammar, in: J. Groenendijk & M. Stokhof (eds), *Amsterdam papers in formal grammar II, Proceedings of the second Amsterdam colloquium on Montague grammar and related topics*, Amsterdam papers in formal grammar 2, Centrale Interfaculteit Univ. of Amsterdam, 1978, pp.211-234.

MONTAGUE, R., 1970, 'Universal grammar', reprinted in: R.H. Thomason (ed.), *Formal Philosophy*, Yale University Press, New Haven, 1974, pp.222-246.

MONTAGUE, R., 1973, 'The proper treatment of quantification in ordinary English', reprinted in: R.H. Thomason (ed.), *Formal Philosophy*, Yale University Press, New Haven, 1974, pp.247-270.

PARTEE, B.H., 1973, 'Some transformational extensions of Montague grammar', reprinted in: B.H. Partee (ed.), *Montague Grammar*, Academic Press, New York, 1976, pp.51-76.

PETRICK, S.R., 1965, *A recognition procedure for transformational grammars*, Ph.D. Thesis, MIT, Cambridge, Mass.