stichting
mathematisch
centrum

MC

AFDELING ZUIVERE WISKUNDE
(DEPARTMENT OF PURE MATHEMATICS)

ZW 99/77 JUNI

T.M.V. JANSSEN

A COMPUTER PROGRAM FOR PTQ AND ITS LOGICAL REDUCTION RULES

Preprint

2e boerhaavestraat 49 amsterdam

Printed at the Mathematical Centre, 49, 2e Boerhaavestraat, Amsterdam.

AMS(MOS) subject classification scheme (1970): 02C10, 68A30, 02C99

A computer program for PTQ and its logical reduction rules [*)]

by

T.M.V. Janssen

ABSTRACT

This paper deals with a computer program that follows the proposals presented by R. MONTAGUE in his article "The Proper Treatment of Quantification in Ordinary English". The problems which arose during the design of the algorithm are discussed. Special attention is paid to the logical reduction rules which were needed to simplify the formulas of intensional logic. An explicit list of these rules is presented, as well as a strategy for their application. The correctness of these rules is formally proved. Finally, several illustrative examples of generated sentences are considered and some inaccuracies and omissions in Montague's article are signalised.

KEY WORDS & PHRASES: *Montague Grammar, Intensional Logic, Computational Linguistics*

[*)] This report will be submitted for publication elsewhere

1. INTRODUCTION*

This contribution deals with a computer program that follows the proposals presented in the article "The Proper Treatment of Quantification in Ordinary English" (MONTAGUE (1973)). This article will be referred to as "PTQ", page numbers are taken from THOMASON (1974). We first will consider a survey of the main parts of the program.

The program is a generating program. It generates syntactic structures according to the syntactic rules in PTQ. Such a structure is a labelled tree resembling those presented in PTQ. On the one hand, the sentence corresponding to this structure is formed; on the other hand each structure is translated into the corresponding formula from intensional logic. Furthermore this formula is reduced in order to obtain a simplified formula, resembling the formula given in PTQ. This whole process is indicated in the following scheme.

$$\begin{tikzcd}
 & \text{formation} \arrow[d, "3"] & \text{generation} \arrow[d, "2{,}4"] & \text{translation} \arrow[d, "3"] & & \text{reduction} \arrow[d, "5"] & \\
\text{sentence} & \Leftarrow & \text{syntactic structure} & \Rightarrow & \text{formula} & \Rightarrow & \text{simplified formula}
\end{tikzcd}$$

In the next sections several parts of the program will be considered; the numbers in the scheme above indicate in which section those parts of the program are dealt with. After that a list is presented of the reduction rules used (section 6), and their correctness is proved in sections 9 and 10. These rules are illustrated by several examples which were generated by the computer (section 11). Some of the generated sentences brought to light inaccuracies or omissions in PTQ.

I will try to indicate the essence of the algorithms and provide motivation for their design. In doing so, I will speak about the computer in rather antropomorphic terms, like "he chooses" or "he wishes"; needless to say, this has nothing to do with reality.

2. GENERATION

In this section will be demonstrated how the computer generates syntactic structures; one aspect of this demonstration will be revised in section 4. The computer generates according to rules S1,...S14 from PTQ. For convenience two of them are indicated:

S4: *if* $\alpha \in P_T$ *and* $\beta \in P_{IV}$ *then* $F_4(\alpha,\beta) \in P_t$

S5: *if* $\alpha \in P_{TV}$ *and* $\beta \in P_T$ *then* $F_5(\alpha,\beta) \in P_{IV}$

As you notice, the rules are formulated in some function-like notation; a corresponding terminology is used: α is called the first argument of the rule, β is called the second argument and F_4 and F_5 are called (string) formation functions.

The computer wishes to make a sentence. He knows several instructions which tell him how a sentence could be formed, e.g. a sentence could be the conjunction of two sentences, *necessarily* followed by a sentence, or formed according to rule S4. The computer makes at random a choice from these instructions; say S4. According to this rule he has to make as the first argument a (member of the category) term (T), and as the second argument an intransitive verb (IV). There are several instructions which tell him how a term could be formed. One is: take a lexical element of the category T. Assume he chooses to do so and takes *Mary*. One of the instructions for making an IV is S5. In this case he has to make as first argument a transitive verb (TV), and as second one a term. Suppose in both cases he chooses to take a lexical element; e.g. *love* and *John*. In this way he has formed the syntactic structure corresponding to the sentence *Mary loves John* (see fig. 1),

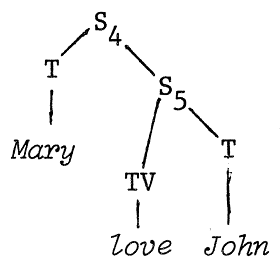
As you notice, making an expression of some category involves making expressions of other categories: the arguments of the chosen rule. For each category the way things go is in essence the same. So the "natural" way to describe this process is by means of a recursive procedure. The language used to write the program in is ALGOL-60; this language (as distinct from FORTRAN) allows for writing recursive procedures.

The kernel of the generating part of the program will be indicated below. The symbol " := " should be read as "becomes", it means that the expression to the left of it is determined by the expression to the right.

In a construction with brackets, e.g. *make (category)*, the expression outside the brackets can be considered as an operator and the expression between the brackets as its argument. *category* is a variable which may be replaced by any specific category.

```
procedure make (category)
begin     rule := choose rule for (category)
        if rule is not take lexical element
        then begin make (argument 1 of (rule));
                  if has two arguments (rule)
                  then make (argument 2 of (rule))
             end
        else choose lexical element of (category)
end
```

figure 1



3. FORMATION AND TRANSLATION

The syntactic structure is some internal data structure, figure 1 gives a graphical representation of the main aspects of such a structure. We wish to obtain as output of the program the sentence corresponding to this structure. In fact, to each vertex (= node) of the tree there corresponds a string (and the sentence corresponds to the root). Such a string is formed by some combination of the strings corresponding to the arguments (except for the lexical elements). The formation of the string is effected, just as in section 2, by a recursive procedure. Some parts of the prodedure (the instructions for S4 and S5) are as follows.

```
procedure form string (vertex)
begin do instruction corresponding to (rule mentioned at (vertex));
  instruction F4: begin form string (argument 1 of (vertex));
                        replace first verb in (form string (argument 2
                        of (vertex)))
                  end
  instruction F5: begin replace in some cases he by him
                        in (form string (argument 1 of (vertex)));
                        form string (argument 2 of (vertex))
                  end
end
```

In order to obtain a visual representation of the tree, the strings corresponding to the lower nodes of the tree are also printed. The output corresponding to figure 1 is presented below. In the remaining part of this paper all syntactic structures will be indicated in this way. In general the formed category and the used syntactic rules are indicated

```
S1:TERM:       Mary
S1: TV :           love
S1:TERM:           John
S5: IV :       love John
S4:SENT:  Mary loves John
```

The syntactic structure has to be translated into a formula of intensional logic. Corresponding to each syntactic rule there is a translation instruction. All these instructions are of the following type:
in order to make the translation, you must first make the translations of the arguments and combine then in a certain way. Therefore, the essence of the translation part is again a recursive procedure. Note that "making the translation of" is indeed a function since it is defined on structures instead of on strings.

English words like *walk* are distinguished from logical constants by a prime' (as in *walk'*). Instead of the constant *j* from PTQ we will use *John'*. The translation of a structure will be presented in the same way as the syntactic structures. As an example the output of the translation corresponding to figure 1 is presented.

$Mary'^{*}$

$love'$

$John'^{*}$

$love'(^{\wedge}John'^{*})$

$Mary'^{*}(^{\wedge}[love'(^{\wedge}John'^{*})])$

4. PROBLEMS WITH HE$_i$

The generation process as described in section 2 leads to a problem with respect to the rule for Term-substitution; this is rule S14,n (*if* $\alpha \in P_T$ *and* $\phi \in P_t$ *then* $F_{10,n}(\alpha,\phi) \in P_t$). A structure that is obtained by using an instance of this rule can be partially indicated as follows

TERM:	*a unicorn*
SENT:	*Mary seeks him*$_1$
S14,1:SENT:	*Mary seeks a unicorn*

This sentence could indeed be generated by the computer with this structure: S14 is one of the instructions and *he*$_1$ is a possible choice for a term. However it should be noticed that also the string *Mary seeks him*$_1$ can be the final result of the generation process. *Him*$_1$, however, is not an English word, and thus the computer would have produced a string which is not a correct English sentence. On the other hand, the computer could have choosen another term instead of *him*$_1$ in the structure presented above, e.g. *John*. In that case *a unicorn* has to be substituted for the first occurrence of *he*$_1$ or *him*$_1$ in the string *Mary seeks John*. There is no such occurrence, so the substitution has no effect. This results in a sentence, with an absurd syntactic structure (and also in an absurd logical structure). A related problem arises with respect to *such that* constructions; these are made by rule S3,n(*if* $\zeta \in P_{CN}$ *and* $\phi \in P_t$ *then* $F_{3,n}(\zeta,\phi) \in P_{CN}$). In case ϕ does not contain an occurrence of *he*$_n$, an incorrect logical interpretation may appear. This is demonstrated by the following example (due to STOKHOF & GROENENDIJK (1976)).

S1 :TERM: *Mary*
S1 : CN : *woman*
SENT: *he*$_1$ *walks*
S3,2 : CN : *woman such that he*$_1$ *walks*
⋮
SENT: *he*$_1$ *loves the woman such that he*$_1$ *walks*
S14,1:SENT: *Mary loves the woman such that she walks*

In the final sentence *she* must refer to *woman*. The presented structure, however, would imply that *she* refers to *Mary*.

In order to avoid all these problems, we will require a "nice" correspondence between occurrences of *he*$_n$ and the rules S3,n and S14,n. Let us call these rules "he$_n$-binding rules", and the second argument of them the "scope of the rule". Now we require that whenever a he$_n$-binding rule is used in the syntactical structure, then there is at least one occurrence of *he*$_n$ within the scope of this rule. Moreover, if there is an occurrence of *he*$_n$ in the structure, then it is within the scope of an he$_n$-binding rule.

This correspondence between the use of he$_n$-binding rules and occurrences of *he*$_n$, makes the choice of a term dependent on the whole syntactic structure. Therefore we must change the generation process. Besides, for technical reasons it is convenient to use for each index n at most once a he$_n$-binding rule (see also section 9). The stages of the generation process are now as follows.

1) Generate the whole syntactic structure without the lexical elements. So the *else* part in the procedure *make* (section 2) must be removed.
2) Insert the terms *he*$_n$. Within the scope of a he$_n$-binding rule there must be at least one insertion of *he*$_n$; a *he*$_n$ may not be inserted outside the scope of such a rule.
3) Insert the words.

5. REDUCTION PRINCIPLES

A formula that is obtained by translating a complete syntactic structure may be fairly complex. The program contains instructions for reducing formulas in order to obtain simplified formulas like the ones presented in PTQ. Such an instruction (=rule) is of the form: under certain conditions, replace a subformula by another one. A reduction rule should transform a formula into a logically equivalent one. For some rules, this needs to be proved. These proofs are presented in sections 9 and 10 in this paper. The main principle of the reduction process is: apply every instruction that can be applied and stop when none can be applied any more. If one actually translates "by hand" one will already during translation simplify the intermediate results. The justification for this way of working is given in section 8. In fact, the computer does his job in this way too, but for perspicuity of exposition I treat the reduction part as a separate stage that starts after the whole translation has been made.

We want the computer to manipulate formulas. For this purpose it is convenient to take for a formula not a string, but a labelled tree. Consider the formula $\lambda x\ {}^{\vee\wedge} man'(x)$. This formula is split up in tree parts: the main operator λ, the variable x, and the remaining part of the formula. That part is split up in the operator ${}^{\vee}$ and the formula ${}^{\wedge} man'(x)$. This formula is called the argument of the operator ${}^{\vee}$. This, in its turn is, also split up. The tree thus obtained is sketched in figure 2.

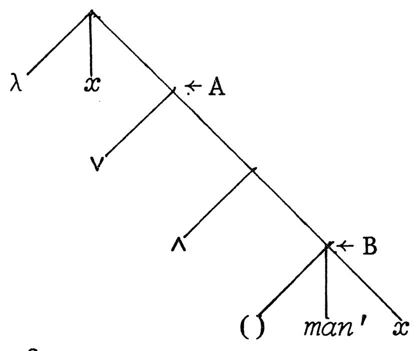


figure 2

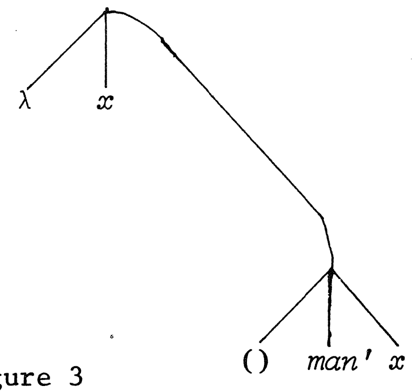


figure 3

One of the reduction rules allows us to replace ${}^{\vee\wedge}\alpha$ by α, thus to replace the formula $\lambda x\ {}^{\vee\wedge} man'(x)$ by $\lambda x(man'(x))$. Since formulas are considered as trees, this involves a tree transformation. The edge (in figure 2) connecting the root of the tree with A must be replaced by an edge connecting the root with B. So we obtain the tree sketched in figure 3. This replacement is produced in the program by an instruction such as *replace* (A,B). The condition for application of this transformation is that there is some operator ${}^{\vee}$ followed by the operator ${}^{\wedge}$. This condition is "local", it can be verified by inspecting a little part of the tree. The change is also local: it consists of replacements of some connections in a rather small part of the tree. The program only contains rules of this "simple" kind (see also section 11).

A fragment of the reduction part is indicated below. It is (again) a recursive procedure. Only the reduction rule treated above is presented. The program tries to apply this rule in a top-down order.

```
procedure reduce (vertex)
begin
    if operator of (vertex) = extension
      and operator of (argument of (vertex)) = intension
    then replace (vertex, argument of (argument of (vertex)))
    else begin reduce (argument 1 of (vertex));
               if has two arguments (rule mentioned at (vertex))
               then reduce (argument 2 of (vertex))
         end
end
```

6. REDUCTION RULES

Three types of reduction rules can be distinguished. These are (I) notational conventions, (II) rules which are true in all models and (III) rules based upon meaning postulates (MP's). For each rule is indicated where its justification can be found: for instance "p.259,-16" refers to THOMASON (1974), page 259, the 16th line from below (a "+" would mean from the top) and "th.8.2" indicates the theorem in section 8 part 2. Some rules are accompanied by remarks or conditions for their application.

Number of rule; formula 1 is replaced by formula 2; type; motivation is on

(R1) c^* $\lambda P[P\{{}^\wedge c\}]$ I p.260,+5

(R2) ${}^\wedge\psi\{\eta\}$ $\psi(\eta)$ I p.259,-5

The notational convention for braces states that $\phi\{\eta\} \equiv {}^\vee\phi(\eta)$. Reduction rule 7 is applied in order to obtain the above formulation

(R3) $\phi(\psi)(\eta)$ $\phi(\eta,\psi)$ I p.259,-8

Condition in PTQ: $\phi(\psi)(\eta)$ is a well formed expression of type t. In section 11 we will see that this condition turns out to be too general. The computer applies this rule when ϕ is the translation of some verb.

(R4) $\delta({}^{\wedge}u)$ $\delta_*(u)$ I p.265,+10

Condition: δ is the translation of an intransitive verb or of a common noun

(R5) $\delta({}^{\wedge}u,{}^{\wedge}[v]^*)$ $\delta_*(u,v)$ I p.265,+10

Condition: δ is the translation of a transitive verb.

(R6) $\delta({}^{\wedge}u,{}^{\wedge}\lambda PP\{{}^{\wedge}v\})$ $\delta_*(u,v)$ I

This is alternative formulation for reduction rule 5 (with the same condition as in (5))

(R7) ${}^{\vee\wedge}\phi$ ϕ II th.10.1

(R8) $\lambda z[..z..](\alpha)$ $[..\alpha..]$ II section 9

Conditions for application are found in section 9.

(R9) $\neg\neg\phi$ ϕ II proof is evident

(R10) $\Box\Box\phi$ $\Box\phi$ II proof is evident

(R11) $\delta(x)$ $\delta_*({}^{\vee}x)$ III p.265,-17,(MP3)

Condition: δ is the translation of an intransive verb other than *rise or change.*

(R12) $\delta(x,P)$ $P\{{}^{\wedge}\lambda y[\delta_*({}^{\vee}x,{}^{\vee}y]\}$ III p265-14 (MP4)

Condition: δ is the translation of a transitive verb other than *seek* or *conceive.*

(R13a) $\bigvee x[\delta(x)\wedge P\{x\}]$ $\bigvee u[\delta({}^{\wedge}u)\wedge P\{{}^{\wedge}u\}]$ III th.10.8

(R13b) $\bigvee x[\delta(x)\wedge[..x..]\wedge P\{x\}]$ $\bigvee u[\delta({}^{\wedge}u)\wedge[..u..]\wedge P\{{}^{\wedge}u\}]$

III r.10.12

Condition for application of R13a or R13b: δ is the translation of a common noun other than *price* or *temperature.* These rules are based upon meaning postulate 2. They allow a variable to be replaced by a variable of another type. Variant R13b is needed when a relative clause is attached to the common noun.

(R14a) $\bigwedge x[\delta(x)\to P\{x\}]$ $\bigwedge u[\delta({}^{\wedge}u)\to P\{{}^{\wedge}u\}]$ III th.10.9

(R14b) $\bigwedge x[\delta(x)\wedge[..x..]\to P\{x\}]$ $\bigwedge u[\delta({}^{\wedge}u)\wedge[..u..]\to P\{u\}]$

III r.10.12

The conditions for application of R14a or R14b are the same as the conditions for rule 13; these rules play a similar role.

(R15a) $\bigvee y\bigwedge x[\delta(x)\leftrightarrow x=y]\wedge P\{y\}]$ $\bigvee v[\bigwedge u[\delta({}^{\wedge}u)\leftrightarrow u=v]\wedge P\{{}^{\wedge}v\}]$

III th.10.10

(R15b) $\bigvee y\bigwedge x[\delta(x)\wedge[..x..]\leftrightarrow x=y]\wedge P\{y\}]$ $\bigvee v[\bigwedge u[\delta({}^{\wedge}u)\wedge[..{}^{\wedge}u..]\leftrightarrow u=v]\wedge P\{{}^{\wedge}v\}]$

III r.10.12.

The conditions and the comment are the same as for R13.

(R16) $in'(P)(Q)(x)$ $P\{{}^{\wedge}\lambda y[in'_{*}({}^{\vee}y)(Q)(x)]\}$ III p.264,+1(MP8)

The rule is formulated so as to parallel the formulations of meaning postulates 3 and 4 in reduction rules 11 and 12. There is a difference: the extension operator is applied to only one variable.

(R17) δ $\lambda y\lambda x\delta(x,y)$ II R3 & evident proof

This rule constitutes an exception to the principle that each rule is applied whenever this is possible. The computer tries to apply R17 if the following 3 conditions are satisfied: (a) the whole sentence has been translated, (b) no rule among R1..R16 applies and (c) the expression in which δ occurs is not of the form $\delta(\alpha,\beta)$.

In PTQ a set of English sentences together with formulas which represent their respective meanings is presented. The above list is intended to contain the reduction rules which are required for deriving these formulas. This intention is in one case not fulfilled (see section 11 example 9). On the other hand R16 goes further than required, since PTQ has no examples with *in*.

Some of the rules might look a little surprising (R13 and R14), others are quite common (R9 and R10). Several are mentioned in the examples in PARTEE (1975); the rules of type II can be found (without proofs) in GALLIN (1976). But no one had so far presented a list of rules needed for PTQ. When working "by hand" one usually has some intuition about what a correct and succesful step for further simplification would be. But the computer needs an list of rules and a strategy for applying them. So programming forced us to make our intuitions explicit.

Moreover we had to be sure that the result is correct (namely an equivalent logical expression). The principle that the correctness of the reduction steps has to be proven is observed by PARTEE (1975). But she only proves correctness for the specific sentences she treats. We will prove that the rules mentioned above will yield a correct result in all possible situations. Thus programming gave rise to theoretical investigations.

7. DEFINITIONS FOR INTERPRETATION

The proofs for the correctness of the reduction rules will be semantic proofs: they are based upon the interpretation of intensional logic in a model $\mathcal{A} = \langle A, I, J, \leq, F\rangle$. We will therefore speak of formulas and valuations. For instance, we will use phrases like "the function such that its value for the element d is the valuation of α with respect to i,j and g". It is convenient to have a symbolic notation at hand. The mentioned phrase is symbolized by "$\lambda d\ \alpha^{\mathcal{A},i,j,g}$". The metalanguage used, contains the symbols α,ϕ,ψ,η (for formulas); a, a_1 (for elements from A); s (from $A^{I\times J}$); i,k,p(from I); j,l,q(from J); $\exists,\forall$(as quantifiers); $\{,\}$(as brackets) and some symbols which are the same as the ones in intensional logic (as $=,\lambda,(,),\neg$). We will omit the sets from which a, a_1, s, i, j, k and l are taken (e.g. $\exists s$ stands for $\exists s \in A^{I\times J}$).

In the formulas of intensional logic the variables u and v will always be of type e, and x and y of type <s,e>. The variables z and w can be of any type, their type will be indicated by ζ and θ. The expression $[\alpha/z]\phi$ is a notation for the formula that is obtained from formula ϕ by replacing all free occurrences of z by α.

An $\mathcal{A}$-assignment g is a function with as its domain the set of all variables and such that $g(z) \in D_{\zeta,A,I,J}$ (see PTQ p.258,+7 for a full definition). Let d be an element from $D_{\theta,A,I,J}$. Then the $\mathcal{A}$-assignment $[w \rightarrow d]g$ is defined by

$$\begin{cases} [w \rightarrow d]g\ (w) = d \\ [w \rightarrow d]g\ (z) = g(z) \text{ if } z \neq w \end{cases}$$

The valuation is a function with parameters i,j and g. Its domain is the set of formulas and its range is $\bigcup_{\zeta} D_{\zeta,A,I,J}$. The <u>valuation</u> of a formula ϕ with respect to i,j and g is written as $\phi^{\mathcal{A},i,j,g}$ and it is defined by the following recursive definition.

(1) $c^{\mathcal{A},i,j,g} = F(c)(i,j)$ if c is a logical constant

(2) $z^{\mathcal{A},i,j,g} = g(z)$ if z is a variable

(3) $\{\lambda z\psi\}^{\mathcal{A},i,j,g} = \lambda d\{\psi\}^{\mathcal{A},i,j,[z\rightarrow d]g}$ where $d \in D_{\zeta,I,J,A}$

(4) $\{\psi(\eta)\}^{\mathcal{A},i,j,g} = \psi^{\mathcal{A},i,j,g}(\eta^{\mathcal{A},i,j,g})$

(5) $\{\psi=\eta\}^{\mathcal{A},i,j,g} = \begin{cases} 1 & \text{if } \psi^{\mathcal{A},i,j,g} = \eta^{\mathcal{A},i,j,g} \\ 0 & \text{otherwise} \end{cases}$

(6) $\{\neg\psi\}^{A,i,j,g} = \begin{cases} 1 & \text{if } \psi^{A,i,j,g} = 0 \\ 0 & \text{otherwise} \end{cases}$ (similarly for $\wedge, \vee, \rightarrow, \Longleftrightarrow$)

(7) $\{\forall z\phi\}^{A,i,j,g} = \begin{cases} 1 & \text{if } \exists d \in D_{\zeta,A,I,J} \text{ such that } \psi^{A,i,j,[z\rightarrow d]g} = 1 \\ 0 & \text{otherwise} \end{cases}$

(8) $\{\Box\psi\}^{A,i,j,g} = \begin{cases} 1 & \text{if } \forall k,l\ \psi^{A,k,l,g} = 1 \\ 0 & \text{otherwise} \end{cases}$

similarly for $H\psi$ and $W\psi$

(9) $\{^{\wedge}\psi\}^{A,i,j,g} = \lambda k,l\ \psi^{A,k,l,g}$

(10) $\{^{\vee}\psi\}^{A,i,j,g} = \psi^{A,i,j,g}(i,j)$

8. REDUCTION BASIS

THEOREM. *Let* α_1 *and* α_2 *be formulas of the same type as* z_1, *and suppose* (A): $\forall_{i,j,g}\ \alpha_1{}^{A,i,j,g} = \alpha_2{}^{A,i,j,g}$.
Then for all formulas ϕ *it holds that* $\forall_{i,j,g}\ \{[\alpha_1/z]\phi\}^{A,i,j,g} = \{[\alpha_2/z]\phi\}^{A,i,j,g}$

PROOF. By formula induction. First we proof the theorem for the case that ϕ is variable or a constant. Next we prove it for compound formulas under the induction hypothesis that the theorem holds for formulas with lower complexity. The cases we have to consider are the 10 cases from the definition of valuation (section 7). With E7 we refer to the 7th clause of this definition, with IH to the induction hypothesis.

1) $\phi \equiv c$ then $[\alpha_1/z]\ c \equiv c \equiv [\alpha_2/z]c$

2) $\phi \equiv w$ if $w \not\equiv z$ then see 1) else

$$\{[\alpha_1/z]z\}^{A,i,j,g} = \alpha_1{}^{A,i,j,g} \overset{A}{=} \alpha_2{}^{A,i,j,g} = \{[\alpha_2/z]z\}^{A,i,j,g}$$

3) $\phi = \lambda w\psi$ if $w = z$ then $[\alpha_1/z]\lambda w\psi \equiv \lambda w\psi \equiv [\alpha_2/z]\lambda w\psi$, else

$\{[\alpha_1/z]\lambda w\psi\}^{A,i,j,g} = \{\lambda w[\alpha_1/z]\psi\}^{A,i,j,g} \overset{E3}{=} \lambda d\{[\alpha_1/z]\psi\}^{A,i,j,[z\rightarrow d]g} =$

$\overset{IH}{=} \lambda d\{[\alpha_2/z]\psi\}^{A,i,j,[z\rightarrow d]g} \overset{E3}{=} \{[\alpha_2/z]\lambda w\psi\}^{A,i,j,g}$

8) $\phi \equiv \Box\psi$ $\{[\alpha_1/z]\ \Box\psi\}^{A,i,j,g} = 1 \Longleftrightarrow \{\Box[\alpha_1/z]\psi\}^{A,i,j,g} = 1 \Longleftrightarrow$

$\overset{E8}{\Longleftrightarrow} \forall k,l\ \{[\alpha_1/z]\psi\}^{A,k,l,g} = 1 \overset{IH}{\Longleftrightarrow} \forall k,l\{[\alpha_2/z]\psi\}^{A,k,l,g}$

$= 1 \overset{E8}{\Longleftrightarrow} \{[\alpha_1/z]\Box\psi\}^{A,i,j,g} = 1$

9) $\phi \equiv {}^\wedge\psi \quad \{[\alpha_1/z]^\wedge\psi\}^{A,i,j,g} = \{^\wedge[\alpha_1/z]\psi\}^{A,i,j,g} \overset{E9}{=} \lambda k,l\{[\alpha_1/z]\psi\}^{A,k,l,g} =$

$\overset{IH}{=} \lambda k,l\ \{[\alpha_2/z]\psi\}^{A,k,l,g} \overset{E9}{=} \{[\alpha_2/z]^\wedge\psi\}^{A,i,j,g}$

The cases 4 to 7 and 10 are left to the reader

CONSEQUENCE. The basis for the reduction process has been laid. The proofs for the reduction rules are such that we prove the equivalence in the sense of (A) of two formulas. Applying a reduction rule means that a subformula is replaced by its equivalent mentioned in the rule. The justification for the replacement of subformulas is provided for by this theorem. The same idea underlies the way we work "by hand"; a formula is reduced even when the complete context has not yet been formed.

9. LAMBDA CONVERSION

THEOREM. *Let* $\lambda z[\phi](\alpha)$ *be a formula and suppose*

I) *No free occurrence of a variable in* α *becomes bound by substitution of* α *for* z *in* ϕ.

and II) *One of the following conditions holds*

II.1) *the variable* z *does not occur within the scope of* $^\wedge$, $\Box$, H *or* W

or

II.2) $\forall i,j,k,l \quad \alpha^{A,i,j,g} = \alpha^{A,k,l,g}$

then

$\forall i,j\ \{\lambda z[\phi](\alpha)\}^{A,i,j,g} = \{[\alpha/z]\phi\}^{A,i,j,g}$

PROOF. Note that $\{\lambda z[\phi](\alpha)\}^{A,i,j,g} = \{\lambda z[\phi]\}^{A,i,j,g}(\alpha^{A,i,j,g}) =$

$\left[\lambda d\phi^{A,i,j,[z\to d]g}\right](\alpha^{A,i,j,g}) = \phi^{A,i,j,[z\to\alpha^{A,i,j,g}]g}$ (the last equality holds because of the meaning of the expression between [and], see section 7). We prove the theorem by proving that for all i,j and h, where h is an assignment such that III: $h(w) = g(w)$ for all free variables w in α, the following holds: $\phi^{A,i,j,[z\to\alpha^{A,i,j,h}]h} = \{[\alpha/z]\phi\}^{A,i,j,h}$

This is proved by formula induction.

1) $\phi \equiv c$ $\quad c^{A,i,j,[z\to\alpha^{A,i,j,h}]h} = [\alpha/z]c^{A,i,j,h}$

2) $\phi \equiv w$ $\quad$ if $w \not\equiv z$ then see 1), else

$$z^{A,i,j,[z\to\alpha^{A,i,j,h}]h} = \alpha^{A,i,j,h} = \{[\alpha/z]z\}^{A,i,j,h}$$

3) $\phi \equiv \lambda w\psi$. If $w \equiv z$ then $[\alpha/z]\lambda z\psi \equiv \lambda z\psi$; the same holds when w does not occur in ψ. In the remaining case we know that w does not occur in α because of requirement I. Then

$$\{\lambda w\psi\}^{A,i,j,[z\to\alpha^{A,i,j,h}]h} \overset{\text{E3}}{=} \lambda d\psi^{A,i,j,[w\to d]\{[z\to\alpha^{A,i,j,h}]h\}} \overset{\text{A}}{=}$$
$$\lambda d\psi^{A,i,j,[z\to\alpha^{A,i,j,[w\to d]h}]\{[w\to d]h\}} \overset{\text{B}}{=} \lambda d[\alpha/z]\psi^{A,i,j,[w\to d]h} \overset{\text{E3}}{=}$$

$[\alpha/z]\lambda w\psi^{A,i,j,h}$. Equality (A) holds since w does not occur in α. Equality (B) holds since we can apply the induction hypothesis with the assignment $[w\to d]h$ (condition III is satisfied since w does not occur in α).

8) $\phi \equiv \Box\psi$ $\;$ If II.1 holds then $[\alpha/z]\Box\psi \equiv \psi$, else

$$\{\Box\psi\}^{A,i,j,[z\to\alpha^{A,i,j,h}]h} = 1 \overset{\text{E8}}{\Longleftrightarrow} \forall k,l\ \psi^{A,k,l,[z\to\alpha^{A,i,j,h}]h} = 1$$
$$\overset{\text{II.2}}{\Longleftrightarrow} \forall k,l\ \psi^{A,k,l,[z\to\alpha^{A,k,l,h}]h} = 1 \Longleftrightarrow \forall k,l\{[\alpha/z]\psi\}^{A,k,l,h} = 1$$
$$\overset{\text{E8}}{\Longleftrightarrow} \{[\alpha/z]\Box\psi\}^{A,i,j,h} = 1$$

The proofs are similar for $H\psi$ and $W\psi$.

9) $\phi \equiv {}^\wedge\psi$ $\quad$ If II.1 holds then $[\alpha/z]{}^\wedge\psi \equiv {}^\wedge\psi$, else

$$\{{}^\wedge\psi\}^{A,i,j,[z\to\alpha^{A,i,j,h}]h} \overset{\text{E9}}{=} \lambda k,l\ \psi^{A,k,l,[z\to\alpha^{A,i,j,h}]h} =$$
$$= \lambda k,l\ \psi^{A,k,l,[z\to\alpha^{A,k,l,h}]h} = \lambda kl\{[\alpha/z]\psi\}^{A,k,l,h} = \{[\alpha/z]{}^\wedge\psi\}^{A,i,j,h}$$

The cases 4 to 7 and 10 are left to the reader.

CONSEQUENCE: The correctness of the reduction rule for λ conversion.[1]
Reduction rule 8: $\lambda z[\ldots z\ldots](\alpha)$ is replaced by $[\ldots\alpha\ldots]$,
provided that one of the following conditions holds:

1) α is a variable

2) α is a formula of the form ${}^\wedge\eta$

3) α is the translation of a proper name
4) z does not occur within the scope of $^{\wedge}$, $\Box$, H or W.
Correctness can be proved with the previous theorem as follows:
Condition I of the theorem holds if z is a variable that corresponds with a term he_i: the special way in which he_n-substitution takes place, guarantees that we have only one binder for x_n. For other variables we created the same situation as follows: the computer does not simply use variables as p,P ... but indexed ones. Each time a translation or reduction instruction introduces a new instance of e.g. P, the computer takes a new index i and uses P_i. So again there is only one binder for each variable and thus I is satisfied. Posibility 4) is equal to II.1; and if one of the others holds then II.2 is satisfied: for 1) and 2) because of the definition of extension, for 3) because of meaning postulate 1.

10. CORRECTNESS PROOFS

10.1 <u>THEOREM</u>. $\{^{\vee\wedge}\alpha\}^{A,i,j,g} = \alpha^{A,i,j,g}$

<u>PROOF</u>. $\{^{\vee\wedge}\alpha\}^{A,i,j,g} = \{^{\wedge}\alpha\}^{A,i,j,g}(i,j) = \lambda k,l\ \alpha^{A,k,l,g}(i,j) = \alpha^{A,i,j,g}$

10.2 <u>REMARK</u>. It is not in general true that $\{^{\wedge\vee}\alpha\}^{A,i,j,g} = \alpha^{A,i,j,g}$.
A counterexample is as follows[2]. Let the logical constant *Bigboss* be of type <s,e>, and as its valuation for i and j have the individual concept on which the predicate *is the most powerfull man on earth* applies in world i on moment j. Suppose that $Bigboss^{A,i,t_1,g} = \lambda i,j\ Ford^{A,i,j,g}$, and $Bigboss^{A,i,t_2,g} = \lambda i,j\ Bresjnev^{A,i,j,g}$. Now the following holds: $\{^{\wedge\vee}\ Bigboss\}^{A,i,t_2,g}(i,t_1) = \lambda k,l[Bigboss^{A,k,l,g}(k,l)](i,t_1) = Bigboss^{A,i,t_1,g}(i,t_1) = Ford^{A,i,t_1,g}$.
But $Bigboss^{A,i,t_2,g}(i,t_1) = Bresjnev^{A,i,t_1,g}$.

10.3 <u>CONVENTION</u>. With δ is denoted the translation of a common noun other than *price* or *temperature*

10.4 <u>CONVENTION</u>. With $A,i,j,g \models \phi$ is a meant $\phi^{A,i,j,g} = 1$. When A,i,j, and g are clear from the context we write $g \models \phi$.

10.5 <u>THEOREM</u>. *Suppose w, x and $^\wedge u$ are of the same type, and ϕ does not contain binders for or occurrences of x and u. Then* g $\models x = {}^\wedge u$ *implies* g $\models [x/w]\phi \iff$ g $\models [{}^\wedge u/w]\phi$.

<u>PROOF</u>. Because of the definition of valuation the following holds g $\models \lambda w[\phi](x) = \lambda w[\phi]({}^\wedge u)$; and in this situation lambda conversion is allowed.

10.6 <u>THEOREM</u>. *If* $\exists s[x\to s]g \models \delta(x)$ *then* $\exists a[x\to s, u\to a]g \models x = {}^\wedge u$.

<u>PROOF</u>. From meaning postulate 2 and $[x\to s]g \models \delta(x)$it follows that $[x\to s]g \models \mathrm{V}u[x={}^\wedge u]$. This means exactly what we have to prove.

10.7 <u>THEOREM</u>.*Suppose the conditions of theorem* 10.5 *are satisfied. Then* g $\models \Lambda x[x/w]\phi \Rightarrow$ g $\models \Lambda u[{}^\wedge u/w]\phi$ *and* g $\models \mathrm{V}u[{}^\wedge u/w]\phi \Rightarrow$ g $\models \mathrm{V}x[x/w]\phi$

<u>PROOF</u>. $\{s \mid s = \{{}^\wedge u\}^{A,i,j,g}\} \subset \{s \mid s = x^{A,i,j,g}\}$

10.8 <u>THEOREM</u>. $A,i,j,g \models \mathrm{V}x[\delta(x) \wedge P\{x\}] \iff A,i,j,g \models \mathrm{V}u[\delta({}^\wedge u) \wedge P\{{}^\wedge u\}]$

<u>PROOF</u>. $\Leftarrow$: See theorem 10.7

$\Rightarrow$: The left hand side means $\exists s[x\to s]g \models \delta(x) \wedge P\{x\}$

with theorem 10.6 we conclude $\exists a[x\to s, u\to a]g \models x = {}^\wedge u$

now apply theorem 10.5, then $[x\to s, u\to a]g \models \delta({}^\wedge u) \wedge P\{{}^\wedge u\}$

and, from definition of valuation $[x\to s]g \models \mathrm{V}u[\delta({}^\wedge u) \wedge P\{{}^\wedge u\}]$

finally, since x does not occur, $g \models \mathrm{V}u[\delta({}^\wedge u) \wedge P\{{}^\wedge u\}]$

10.9 <u>THEOREM</u>. $A,i,j,g \models \Lambda x[\delta(x) \to P\{x\}] \iff A,i,j,g \models \Lambda u[\delta({}^\wedge u) \to P\{{}^\wedge u\}]$

<u>PROOF</u>. $\Rightarrow$: apply theorem 10.7

$\Leftarrow$: by contraposition. Suppose that it was not true that

$g \models \Lambda x[\delta(x) \to P\{x\}]$

this means that $g \models \neg\Lambda x[\delta(x) \to P\{x\}]$

which is equivalent to $g \models \mathrm{V}x[\delta(x) \wedge\neg P\{x\}]$

Now we apply theorem 10.8 and find $g \models \mathrm{V}u[\delta({}^\wedge u) \wedge\neg P\{{}^\wedge u\}]$

therefore it is not true that $g \models \Lambda u[\delta({}^\wedge u) \to P\{{}^\wedge u\}]$

10.10 <u>THEOREM</u>. $A,i,j,g \models \mathrm{V}y[\Lambda x[\delta(x) \leftrightarrow x = y] \wedge P\{y\}] \Rightarrow$
$A,i,j,g \models \mathrm{V}u\Lambda u[\delta({}^\wedge u) \leftrightarrow u = v] \wedge P\{{}^\wedge u\}]$

<u>PROOF</u>. The left hand side is equivalent to
$\exists$s such that I: $[y{\to}s]g \models \Lambda x[\delta(x) \leftrightarrow x = y]$ and II: $[y{\to}s]g \models P\{y\}$
From I follows $[y{\to}s]g \models \delta(y) \leftrightarrow y = y$, so $[y{\to}s]g \models \delta(y)$.
Apply theorem 10.6 $\exists a[y{\to}s,v{\to}a]g \models y = {}^\wedge v$,
Substitution in II (theorem 10.5) gives III: $[y{\to}s,v{\to}a]g \models P\{{}^\wedge v\}$
Substitution in I and theorem 10.7 gives IV: $[y{\to}s,v{\to}a]g \models \Lambda u[\delta({}^\wedge u) \leftrightarrow {}^\wedge u = {}^\wedge v]$
Since $\forall i,j\ \{{}^\wedge u = {}^\wedge v\}^{A,i,j,g} = \{u=v\}^{A,i,j,g}$ we may replace
${}^\wedge u = {}^\wedge u$ by $u = v$ accordingly to section 8. The combination of III with the result on IV of this replacement gives the right hand side.

10.11 <u>THEOREM</u>. $A,i,j,g \models \mathrm{V}v[\Lambda u[\delta({}^\wedge u) \leftrightarrow u=v] \wedge P\{{}^\wedge u\}] \Rightarrow$
$A,i,j,g \models \mathrm{V}y[\Lambda x[\delta(x) \leftrightarrow x = y] \wedge P\{y\}]$

<u>PROOF</u>. The left hand side is equivalent to
$\exists$a such that I: $[v{\to}a]g \models \Lambda u[\delta({}^\wedge u) \leftrightarrow u=v]$ and II: $[v{\to}a]g \models P\{{}^\wedge v\}$.
In I we replace $u = v$ by ${}^\wedge u = {}^\wedge v$ and obtain thus I':
Let us suppose that $[x{\to}s]g \models \delta(x)$.
Then (theorem 10.6 and 10.7) $\exists a_1[x{\to}s,w{\to}a_1]g \models \delta({}^\wedge w)$
Use I': $[x{\to}s,w{\to}a_1,v{\to}a]g \models {}^\wedge w = {}^\wedge v$, so $[x{\to}s,w{\to}a_1,v{\to}a]g \models x = {}^\wedge v$
We have proved that $[v{\to}a]g \models \Lambda x[\delta(x) \to x={}^\wedge v]$
On the other hand, suppose $[v{\to}a,x{\to}s]g \models x = {}^\wedge v$
then by I it follows that $[v{\to}a,x{\to}s]g \models \delta({}^\wedge v)$, so $[v{\to}a,x{\to}s]g \models \delta(x)$
Thus we proved $[v{\to}a]g \models \Lambda x[x={}^\wedge v \to \delta(x)]$.
Combination with II and application of theorem 10.7 gives the desired result.

10.12 <u>REMARK</u>. The last four proofs also hold when $\delta(x)$ is replaced by $\delta(x) \wedge [\ldots x \ldots]$. So the correctness of the b version of reduction rules 13,14 and 15 follows.

11. EXAMPLES

In this section I present several examples of the treatment of sentences (or parts thereof) as these have actually been executed by the computer. On the basis of these examples some comments on PTQ and on the program are made.

EXAMPLE 1. Syntax
The following sentence is generated

Mary is her and love John.

This sentence illustrates two syntactic inaccuracies of PTQ. Only the first verb has been conjugated, and *her* occurs instead of *herself*. The sentences generated by the computer reveal several syntactic inaccuracies. For instance, disjunction and conjunction cause trouble in combination with rules S5, S14 and S17. Since PTQ has already been studied for several years, it is not surprising that most of these inaccuracies are known (e.g. BENNETT (1974)). The non-obvious inaccuracies discovered with the help of the computer are rather in the "logical" part of PTQ where the situation is less perspicuous. In checking some versions of GROENENDIJK & STOKHOF (1976) the computer assistence appeared to be also useful for the syntax.

EXAMPLE 2. Two relative clauses
The following is a part of a generated structure.

woman

he$_2$ *loves a woman*

S3,2: *woman such that she loves a woman*

he$_1$ *runs*

S3,1: *woman such that she loves a woman such that she runs*

The last string can only be taken to be about a woman which loves a running woman. The structure however indicates that there are two relative clauses specifying a single head noun. The translation of the example above mentions one loving and running woman and not, as is desired, a woman loving a running woman. Thus we note that applying the same syntactic rule several times in succession, can lead to incorrect results. The relation to the head noun, however, is correct in the following example (due to J. Bresnan, mentioned in PARTEE (1975)): *Every girl who attended a womans college who made a donation to it was included in the list.*

EXAMPLE 3. A simple case

	Syntactic Structure		Translation & Reduction
S1:TERM:	*Mary*	T1d:	$Mary$
S1: IV :	*run*	T1a:	run'
S4:SENT:	*Mary runs*	T4:	$Mary'^{*}({}^{\wedge}run')$
		R1:	$\lambda P\, P\{{}^{\wedge}Mary'\}({}^{\wedge}run)$
		R8:	${}^{\wedge}run'\ \{{}^{\wedge}Mary'\}$
		R2:	$run'\ ({}^{\wedge}Mary')$
		R4:	$run'_{*}\ (Mary')$

This derivation is completely regular. The notational convention R4 has been used to introduce the $_{*}$. Instead of R4, also R11 was applicable; the choice depends on the order of the reduction instructions in the program. The computer tries to apply the notational conventions first of all. The sentence *Ninety rises* could also be generated with the same structure. Translation and reduction proceeds in exactly the same way: $rise'\ ({}^{\wedge}ninety')$ is obtained and next R4 is applied. Thus we note that the formula in PTQ (p.268,+10) is not the final one.

EXAMPLE 4. Adverbs

	Syntactic Structure		Translation & Reduction
S1:TERM:	*John*	T1d:	$John'^{*}$
S1: IAV:	*slowly*	T1a:	$walk'$
S1: IV :	*walk*	T1a:	$slowly'$
S10: IV:	*walk slowly*	T10:	$slowly'\ ({}^{\wedge}walk')$
S4:SENT:	*John walks slowly*	T4:	$John'^{*}\ ({}^{\wedge}[slowly'({}^{\wedge}walk')])$
		R1:	$\lambda P\, P\{{}^{\wedge}John'\}({}^{\wedge}[slowly'({}^{\wedge}walk')])$
		R8:	${}^{\wedge}[slowly'({}^{\wedge}walk')]\ \{{}^{\wedge}John'\}$
		R2:	$slowly'({}^{\wedge}walk')\ ({}^{\wedge}John')$
		R3:	$slowly'({}^{\wedge}John', {}^{\wedge}walk')$

We notice that *slowly* is treated as a two place relation. This is clearly not intended in PTQ. If expressions of the category IAV are relations then so is *about a unicorn* in *John talks about a unicorn*. In PTQ (p.267,+13) we see that this is not meant. For this reason we have in the program as condition for R3 not "$\gamma(\alpha)(\beta)$ is a meaningful expression of type t" but rather "γ translates a verb and for this verb R3 has not yet been applied".

As a result the computer does not perform this reduction step in unintended situations as above.

<u>EXAMPLE 5</u>. Seek$_*$

	Syntactic Structure		Translation with immediate reduction
CN:	*unicorn*	T1a:	$unicorn'$
S2:	*a unicorn*	T2 :	$\lambda P_1[Vx_2[unicorn'(x_2) \wedge P\{x_2\}]]$
T:	*Mary*	R13:	$\lambda P_1[Vu_2[unicorn'(^\wedge u_2) \wedge P\{^\wedge u_2\}]]$
IV:	*seek*	R4:	$\lambda P_1[Vu_2[unicorn'_*(u_2) \wedge P\{^\wedge u_2\}]]$
T:	*he$_1$*	T1d:	$Mary'*$
S5:	*seek him$_1$*	R1:	$\lambda P_2[P_2\{^\wedge Mary'\}]$
S4:	*Mary seeks him$_1$*	T1d:	$seek'$
S14,1:	*Mary seeks a unicorn*	T1e:	$\lambda P_3[P_3\{x_1\}]$
		T5 :	$seek'(^\wedge\lambda P_3[P_3\{x_1\}])$
		T4:	$\lambda P_2[P_2\{^\wedge Mary'\}](^\wedge[seek'(^\wedge\lambda P_3[P_3\{x_1\}])])$
		R8:	$^\wedge[seek'(^\wedge\lambda P_3[P_3\{x_1\}])]\{^\wedge Mary'\}$
		R2:	$seek'(^\wedge\lambda P_3[P_3\{x_1\}])\ (^\wedge Mary')$
		R3:	$seek'(^\wedge Mary', ^\wedge\lambda P_3[P_3\{x_1\}])$
		T14:	$\lambda P[Vu_2[unicorn'_*(u_2) \wedge P\{^\wedge u_2\}]](^\wedge\lambda x_1[seek'(^\wedge Mary', ^\wedge\lambda P_3[P_3\{x_1\}])])$
		R8:	$Vu_2[unicorn'_*(u_2) \wedge\ ^\wedge\lambda x_1[seek'(^\wedge Mary', ^\wedge\lambda P_3\ P_3\{x_1\})]\{^\wedge u_2\})$
		R2:	$Vu_2[unicorn'_*(u_2) \wedge \lambda x_1[seek'(^\wedge Mary', ^\wedge\lambda P_3\ P_3\{x_1\})](^\wedge u_2)]$
		R8:	$Vu_2[unicorn'_*(u_2) \wedge seek'(^\wedge Mary', ^\wedge\lambda P_3\ P_3\{^\wedge u_2\})]$
		R6:	$Vu_2[unicorn'_*(u_2) \wedge seek'_*(Mary', u_2)]$

In the translation of *a unicorn* the variable x_2 is used since x_1 was reserved for the translation of he_1. When the variable x_2 is replaced, its index, for technical reasons, is kept the same. This example demonstrates how the computer obtains a reduced formula by conscientiously applying the reduction rules. It is an excellent tool for doing such tedious computations.

When the sentence *Mary finds a unicorn* with the same syntactic structure (i.e. with S14,1) is treated, the same reduction steps can be made; so the last step is an application of notational convention R6. But a different process is also possible. After having obtained

$$find'(^\wedge Mary', ^\wedge\lambda P_3[P_3\{x_1\}])$$

we may apply the meaning postulate for find (R12). Then we obtain

$${}^{\wedge}\lambda P_3[P_3\{x_1\}]\{{}^{\wedge}\lambda y_1[find'_*({}^{\vee\wedge}Mary', {}^{\vee}y_1)]\}$$

this reduces (after R2,R8, R2,R8,R7) to

$$find'_*(Mary', {}^{\vee}x_1).$$

This expression has to be combined with the translation of *a unicorn* and after reduction we obtain

$$Vu_2[unicorn'_*(u_2) \wedge find'_*\,(Mary', u_2)].$$

In general it makes no difference in which order we apply the translation or reduction rules. Sooner or later we have to apply the same rules to the same kind of expressions. The rules for introduction of δ_* constitute the exception to this situation. The above considerations on *Mary finds a unicorn* demonstrate this. Once the meaning postulate is applied (and a corresponding sequence of reduction steps is required) we cannot apply the notational convention any more. Happily these two ways always yield the same final result. The computer tries at each stage to reduce as much as possible, so in the above case $find'_*$ is introduced by means of the meaning postulate. Notice that in example 3 we also had two ways of introducing run'_*; the formulas obtained in these ways would be identical.

<u>EXAMPLE 6</u>. Survey of δ_*

	Syntactic Structure		Translation & Reduction
S1: CN :	*price*	T1:	$price'$
S2:TERM:	*a price*	T2:	$\lambda P_1[Vx_1[price'(x_1) \wedge P_1\{x_1\}]]$
S1: IV:	*rise*	T1:	$rise$
S4:SENT:	*A price rises*	T4:	$\lambda P_1[Vx_1[price'(x_1) \wedge P_1\{x_1\}]]({}^{\wedge}rise')$
		R8:	$Vx_1[price'(x_1) \wedge {}^{\wedge}rise'\{x_1\}]$
		R2:	$Vx_1[price'(x_1) \wedge rise'(x_1)]$

The resulting formula does not allow for application of the meaning postulate for intransitive verbs since *rise* is an exception to the postulate, nor can the notational convention be applied since then an argument of the form ${}^{\wedge}u_1$ is required.
If the sentence had been

A price runs

we could apply the meaning postulate since run' is not an exception. Then we obtain

$$\bigvee x_1[price'(x_1) \wedge run'_*({}^\vee x_1)].$$

If the sentence had been

A unicorn rises

then we might apply the meaning postulate for common nouns (R13), thus obtaining

$$\bigvee u_1[unicorn'_*(u_1) \wedge rise'({}^\wedge u_1)].$$

Next the notational convention can be applied to *rise'*; obtaining

$$\bigvee u_1[unicorn'_*(u_1) \wedge rise'_*(u_1)].$$

Finally we might consider the sentence

A unicorn runs.

Here we have two ways of introducing run'_* (see example 3): the meaning postulate and the notational convention.

<u>EXAMPLE 7</u>. In

	Syntactic Structure
S1 :TERM:	*Bill*
S1 :PREP:	*in*
S1 : CN :	*park*
S2 :TERM:	*the park*
S6 : IAV:	*in the park*
S1 : IV :	*walk*
S10: IV :	*walk in the park*
Su :SENT:	*Bill walks in the park*

The translation of *Bill* reduces (after R1) to

$$\lambda P_1 P_1\{{}^\wedge Bill'\}$$

For the expression *in the park* we obtain (after R15,R4)

$$in'({}^\wedge\lambda P_2[\bigvee v_1[\bigwedge u_1[park'_*(u_1) \leftrightarrow u_1 = v_1] \wedge P_2\{{}^\wedge v_1\}]])$$

So for *Bill walks in the park* we obtain

$$\lambda P_1[P_1\{{}^\wedge Bill'\}]({}^\wedge in'({}^\wedge\lambda P_2[\bigvee v_1[\bigwedge u_1[park'_*(u_1) \leftrightarrow u_1 = v_1] \wedge P_2\{{}^\wedge v_1\}]])({}^\wedge walk')])$$

After application of R8 and R2 this becomes

$$in'({}^\wedge\lambda P_2[\bigvee v_1[\bigwedge u_1[park'_*(u_1) \leftrightarrow u_1 = v_1] \wedge P_2\{{}^\wedge v_1\}]])({}^\wedge walk')({}^\wedge Bill')$$

Next R16 is applied: the meaning postulate for *in*:

$$^\wedge\lambda P_2[\bigvee v_1[\bigwedge u_1[park'_*(u_1) \leftrightarrow u_1=v_1]\wedge P_2\{^\wedge v_1\}]])\{^\wedge\lambda y_2[in'_*(^\vee y_2)(^\wedge walk')(^\wedge Bill')]\}$$

After R2, R8, R2, R8, R7 we obtain

$$\bigvee v_1[\bigwedge u_1[park'_*(u_1) \leftrightarrow u_1=v_1] \wedge in'_*(v_1)(^\wedge walk')(^\wedge Bill')].$$

In PTQ no examples concerning the meaning postulate for *in* are mentioned. This example illustrates the consequence of the meaning postulate: if someone walks in "an individual concept", there is a corresponding "individual" in which he walks.

EXAMPLE 8. Try to find

We consider the sentence

John tries to find a unicorn.

Let us assume that term substitution (S14,i) was not used in the generation process. The translation of *find a unicorn* becomes (after some reductions):

$$find'(^\wedge\lambda P_2[\bigvee u_1[unicorn'_*(u_1) \wedge P_2\{^\wedge u_1\}]]).$$

Since the translation of *John* becomes $\lambda P_1[P_1\{^\wedge John'\}]$ we obtain for the translation of the whole sentence

$$\lambda P_1[P_1\{^\wedge John'\}](^\wedge try\ to'(^\wedge find'(^\wedge\lambda P_2[\bigvee u_1[unicorn'_*(u_1)\wedge P_2\{^\wedge u_1\}]]))).$$

After application of R8,R2 and R3 we obtain

$$try\ to'(^\wedge John', ^\wedge[find'(^\wedge\lambda P_2[\bigvee u_1[unicorn'_*(u_1)\wedge P_2\{^\wedge u_1\}]])]).$$

No one of the reduction rules R1-R16 applies any more, so the computer now applies R17 and obtains

$$try\ to'(^\wedge John, ^\wedge[\lambda y_1\lambda x_2[find'(x_2,y_1)](^\wedge\lambda P_2[\bigvee u_1[unicorn'_*(u_1)\wedge P_2\{^\wedge u_1\}]])]).$$

Since *find'* now has become a two-place relation, the meaning postulate can be applied to it and after application of R8 and R2 several times (and R7 once) the final result is obtained:

$$try\ to'(^\wedge John, ^\wedge\lambda x_2[\bigvee u_1[unicorn'_*(u_1)\wedge find'_*(^\vee x_2,u_1)]])$$

Our reduction principle is "at each stage: apply reduction rules until none applies any more". Rule R17 is an exception to this principle. On the one hand: treating R17 as an "everywhere" rule (which is more elegant than our conditions for application) would almost be the same as translating *find* by $\lambda y\lambda x\ find'(x,y)$. This is, however, not done in PTQ. Moreover we would in that case apply R17 in many cases were this is not needed (in the previous examples we never used this rule). On the other hand; without R17 we would end up with a formula which deviates from the one given in PTQ[3].

EXAMPLE 9. Program Failure

Generated sentence: *Mary is a woman*

Reduced formula : $\vee u[woman'_*(u) \wedge Mary'=u]$

In this situation we could do one traditional reduction step more and thus obtain $woman'_*(Mary')$. It is a shortcoming of the program that it does not account for this last step. The background for this is as follows. All the reduction rules are local (as explained in section 6). The reduction step under discussion however is not local. In order to decide whether a subformula $u \equiv Mary'$ implies that all occurrences of u may be replaced by $Mary'$, the whole formula must be taken into account. If we had the connective "$\rightarrow$" instead of "$\wedge$" then we could not reduce the formula any further. To handle this case, we need a reduction rule of a new, complex kind. The same need arises in the following situation.

Generated sentence: *The fish loses the fish*

Reduced formula: $\vee v_1[\wedge u_1[fish'_*(u_1) \leftrightarrow u_1 = v_1] \wedge \vee v_2[\wedge u_2[fish'_*(u_2) \leftrightarrow u_2=v_2] \wedge lose'_*(v_1,v_2)]]$

This formula expresses that there is exactly one fish and that, this fish loses itself. The formula is equivalent to $\vee v_1[\wedge u_1[fish'_*(u_1) \leftrightarrow u_1=v_1] \wedge lose'_*(v_1,v_1)]]$. Again, to make this step a non local verification must be made.

It would of course be possible to give two rules, treating these incidental problems: That would be an ad hoc solution and other constructions can easily be envisaged where non-local rules would be needed.

A solution to this kind of problems seems difficult to obtain, needing powerful innovations. Moreover, one may only expect to obtain partial solution to the reduction problem. Since intensional logic is undecidable, a terminating program reducing all formulas of intensional logic to a "simplest" form cannot exist.

REFERENCES

BENNET, M.R., 1974, *Some extensions of a Montague Fragment of English*, Ph.D. thesis UCLA, University Microfilms. (also published by the University of Indiana Linguistics Club).

GALLIN, D., 1975, *Intensional and higher-order modal logic*, Amsterdam, North-Holland Publishing Company.

GROENENDIJK, JEROEN & STOKHOF, MARTIN 1976, *Some notes on personal pronouns, reflexives and sloppy identity in a Montague Grammar*, In: Akten des 10 Linguistischen Kolloquiums, Band I Grammatik, Tübingen, Niemeyer Verlag.

MONTAGUE, R., 1973, *The Proper Treatment of Quantification in Ordinary English*, Reprinted in THOMASON (1974).

PARTEE, B., 1975, *Montague Grammar and Transformational Grammar*, Linguistic Inquiry 6, pp. 203-300.

THOMASON, R.H., 1974, *Formal Philosophy. Selected Papers of Richard Montague*, New Haven and London, Yale University Press.

NOTES

* This article is a revision of the paper presented at the first Amsterdam Colloquium on Montague Grammar and related topics (January 1976). I am indebted to Johan van Benthem, Peter van Emde Boas, Joyce Friedman, Jeroen Groenendijk, Ewan Klein, Martin Stokhof and Zeno Swijtink for their remarks in several stages of the work on this subject.

1 The need for a complex conversion rule is brought to my attention by Zeno Swijtink.

2 This counter-example is based upon an idea of Johan van Benthem.

3 The use of this rule was brought to my attention by Joyce Friedman (she also works on a computer program for PTQ). Without this rule, the only way to obtain the PTQ formula for *John tries to find a unicorn* was to start with a structure in which S14,i was used: *John tries to* [*a unicorn*[*find him*$_1$]]. This observation is due to Martin Stokhof.