CWI Tract 19

Foundations and applications of Montague grammar
Part 1: Philosophy, framework, computer science

T.M.V. Janssen

Centrum voor Wiskunde en Informatica
Centre for Mathematics and Computer Science

1980 Mathematics Subject Classification: 08A99, 03G15, 68F05, 03B15, 68F20.
1982 CR Categories: F.3.2, I.2.7, F.3.1, F.3.0, F.4.3.
ISBN 90 6196 292 7

PREFACE

The present volume is one of the two tracts which are based on my dissertation 'Foundations and applications of Montague grammar'. Volume 1 consists of the chapters 1,2,3 and 10 of that dissertation, and volume 2 of the chapters 4-9. Only minor corrections are made in the text. I would like to thank here again everyone who I acknowledged in my dissertation, in particular my promotor P. van Emde Boas, co-promotor R. Bartsch, and coreferent J. van Benthem. For attending me on several (printing-)errors in my dissertation I thank Martin van de Berg, Cor Baayen, Biep Durieux, Joe Goguen, Fred Landman and Michael Moortgat, but in particular Herman Hendriks, who suggested hundreds of corrections. The illustrations are made by Tobias Baanders.

The two volumes present an interdisciplinary study between mathematics, philosophy, computer science, logic and linguistics. No knowledge of specific results in these fields is presupposed, although occasionally terminology or results from them are mentioned. Throughout the text it is assumed that the reader is acquainted with fundamental principles of logic, in particular of model theory, and that he is used to a mathematical kind of argumentation. The contents of the volumes have a lineair structure: first the approach is motivated, next the theory is developed, and finally it is applied. Volume 1 contains an application to programming languages, whereas volume 2 is devoted completely to the consequences of the approach for natural languages.

The volumes deal with many facets of syntax and semantics, discussing rather different kinds of subjects from this interdisciplinary field. They range from abstract universal algebra to linguistic observations, from the history of philosophy to formal language theory, and from idealized computers to human psychology. Hence not all readers might be interested to read everything. Readers only interested in applications to computer science might restrict them selves to volume 1, but then they will miss many arguments in volume 2 which are taken from computer science. Readers only interested in applications to natural language might read chapters 1-3 of volume 1, and all of volume 2, but they will miss several remarks about the connection between the study of the semantics of programming languages and of the semantics of natural languages. Readers familiar with Montague grammar, and mainly interested in practical consequences of the approach, might read chapters 1 and 2 in volume 1 and chapters 6-10 in volume 2, but they will miss new arguments and results concerning many aspects of Montague grammar.

Each chapter starts with an abstract. Units like theorems etc. are numbered (eg 2.3 Theorem). Such a unit ends where the next numbered unit starts, or where the end of the unit is announced (2.3 end). References to collected works are made by naming the first editor. Page numbers given in the text refer to the reprint last mentioned in the list of references, except in case of some of Frege's publications (when the reprint gives the original numbering).

CONTENTS

I. The principle of compositionality of meaning 1

1. An attractive principle 2
2. Frege and the principle 5
   - 2.1. Introduction 5
   - 2.2. Grundlagen 6
   - 2.3. Sinn und Bedeutung 8
   - 2.4. The principle 9
   - 2.5. Conclusion 11
3. Towards a formalization 11
4. An algebraic framework 17
5. Meanings 28
   - 5.1. Introduction 28
   - 5.2. Natural language 29
   - 5.3. Programming Language 29
   - 5.4. Predicate Logic 30
   - 5.5. Strategy 34
   - 5.6. Substitutional Interpretation 34
6. Motivation 35

II. The algebraic framework 41

1. Introduction 42
2. Algebras and subalgebras 43
3. Algebras for syntax 50
4. Polynomials 56
5. Term algebras 61
6. Homomorphisms 67
7. A safe deriver 75
8. Montague grammar 81
9. Discussion 90

III. Intensional logic 95

1. Two facets 96
   - 1.1. Introduction 96
   - 1.2. Model-part I 96
   - 1.3. Model-part II 98
   - 1.4. Laws 98
   - 1.5. Method 99
2. Two-sorted type theory 100
3. The interpretation of Ty2 103
4. Properties of Ty2 106
5. Intensional Logic 113
6. Properties of IL 117
7. Extension and intension 123

IV Montague grammar and programming languages 127
- 1. Assignment statements 128
  - 1.1. Introduction 128
  - 1.2. Simple assignments 129
  - 1.3. Other assignments 131
- 2. Semantics of programs 133
  - 2.1. Why? 133
  - 2.2. How? 135
- 3. Predicate transformers 137
  - 3.1. Floyd's forward predicate transformer 137
  - 3.2. Hoare's backward predicate transformer 139
  - 3.3. Problems with Floyd's rule 139
  - 3.4. Predicate transformers as meanings 141
- 4. Semantical Considerations 144
  - 4.1. The model 144
  - 4.2. The logic 148
  - 4.3. Theorems 150
- 5. First fragment 152
  - 5.1. The rules 152
  - 5.2. Examples 154
- 6. Pointers and arrays 156
  - 6.1. Pointers 156
  - 6.2. Arrays 158
- 7. Second fragment 161
  - 7.1. The rules 161
  - 7.2. The postulates 164
  - 7.3. A model 166
- 8. Correctness and completeness 168
  - 8.1. State transition semantics 168
  - 8.2. Strongest postconditions 169
  - 8.3. Completeness 172
- 9. The backward approach 176
  - 9.1. Problems with Hoare's rule 176
  - 9.2. Backward predicate transformers 177
  - 9.3. Weakest preconditions 178
  - 9.4. Strongest and weakest 179
  - 9.5. Correctness proof 182
- 10. Mutual relevance 185

Appendix Safe and polynomial 189

Index of names 193

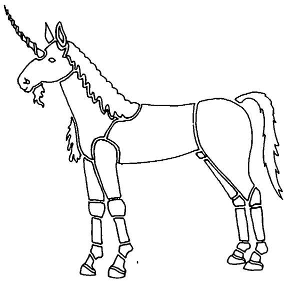
References 197

CHAPTER I

THE PRINCIPLE OF COMPOSITIONALITY OF MEANING

ABSTRACT

This chapter deals with various aspects of the principle of compositionality of meaning. The role of the principle in the literature is investigated, and the relation of the principle to Frege's works is discussed. A formalization of the principle is outlined, and several arguments are given in support of this formalization.



1. AN ATTRACTIVE PRINCIPLE

The starting point of the investigations in this book is the principle of compositionality of meaning. This principle says:

The meaning of a compound expression
is built up from the meanings of its parts.

This is an attractive principle which pops up at a diversity of places in the literature. The principle can be applied to a variety of languages: natural, logical and programming languages. I would not know of a competing principle. In this section the attractiveness of the principle will be illustrated by means of many quotations.

In the philosophical literature the principle is well known, and generally attributed to the mathematician and philosopher Gottlob Frege. An example is the following quotation. It gives a formulation of the principle which is about the same as the formulation given above. THOMASON (1974,p.55) says:

> *Sentences* [..] *such as 'The square root of two is irrational', and 'Two is even',* [..] *ought to be substitutable* salva veritate *in all contexts obeying Frege's principle that the meaning of a phrase is a function of the meanings of its parts.*

Another illustration of the fame of the principle is given by DAVIDSON (1967, p.306):

> *If we want a theory that gives the meaning (as distinct from reference) of each sentence, we must start with the meaning (as distinct from reference) of the parts.*

Next he says:

> *Up to here we have been following Frege's footsteps; thanks to him the path is well known and even well worn.*

Popper mentions a version of the principle which applies to whole theories (POPPER 1976, p.22):

> [..] *the meaning of a theory* [..] *is a function of the meanings of the words in which the theory is formulated.*

Thereafter he says (ibid. p.22):

> *This view of the meaning of a theory seems almost obvious; it is widely held, and often unconsciously taken for granted.*

(For completeness of information: there is, according to Popper, hardly any truth in the principle). Concerning the origin of the principle, Popper remarks in a footnote (ibid. p.198):

> *Not even Gottlob Frege states it quite explicitly, though this doctrine is certainly implicit in his 'Sinn und Bedeutung', and he even produces there arguments in its support.*

In the field of semantics of natural languages, the principle is found implicitly in the works of Katz and Fodor concerning the treatment of semantics in transformational grammars. An explicit statement of the principle is KATZ (1966, p.152):

> *The hypothesis on which we will base our model of the semantic component is that the process by which a speaker interprets each of the infinitely many sentences is a compositional process in which the meaning of any syntactically compound constituent of a sentence is obtained as a function of the meanings of the parts of the constituent.*

Katz does not attribute the principle to Frege; his motivation is of a technical nature (ibid. p.152):

> *Accordingly, we again face the task of formulating an hypothesis about the nature of a finite mechanism with an infinite output.*

The principle is mentioned explicitly in important work on the semantics of natural languages by logicians, and it is related there with Frege. Cresswell develops a mathematical framework for dealing with semantics, and having presented his framework he says (CRESSWELL 1973, p.19):

> *These rules reflect an important general principle which we shall discuss later under the name 'Frege's principle', that the meaning of the whole sentence is a function of the meanings of it parts.*

For another logician, Montague, the principle seems to be a line of conduct (MONTAGUE 1970a, p.217):

> *Like Frege, we seek to do this* [..] *in such a way that* [..] *the assignment to a compound will be a function of the entities assigned to its components* [..].

The principle is implicitly followed by all logic textbooks when they define, for instance, the truth value of $\underline{p} \wedge \underline{q}$ as a function of the truth-values of $\underline{p}$ and of $\underline{q}$. In logic the enormous technical advantages of treating semantics in accordance with the principle are demonstrated frequently. For instance, one may use the power of induction: theorems with a semantic content can be proven by using induction on the construction of the expression under consideration. Logic textbooks usually do not say much about the motivation for their approach or about the principles of logic. In any case, I have not succeeded in finding a quotation in logic textbooks concerning the background of their compositional approach. Therefore, here is one from another source. In a remark concerning the work of Montague, Partee says the following about the role of compositionality in logic (PARTEE 1975, p.203):

> *A central working premise of Montague's theory* [..] *is that the syntactic rules that determine how a sentence is built up out of smaller syntactic parts should correspond one-to-one with the semantic rules that tell how the meaning of a sentence is a function of the meanings*

of its parts. This idea is not new in either linguistics or philosophy; in philosophy it has its basis in the work of Frege, Tarski, and Carnap, and it is standard in the treatment of formalized languages [..].

Since almost all semantic work in mathematical logic is based upon Tarski, mathematical logic is indirectly based upon this principle of compositionality of meaning.

In the field of semantics of programming languages compositionality is implicit in most of the publications, but it is mentioned explicitly only by few authors. In a standard work for the approach called 'denotational semantics', the author says (STOY 1977, pp.12-13):

We give 'semantic valuation functions' which map syntactic constructs in the program to the abstract values (numbers, truth values, functions etc.) which they denote. These valuation functions are usually recursively defined: the value denoted by a construct is specified in terms of the values denoted by its syntactic subcomponents [..].

It becomes clear that this aspect is a basic principle of this approach from a remark of Tennent in a discussion of some proposals concerning the semantics of procedures. Tennent states about a certain proposal the following (NEUHOLD 1978,p.163).

Your first two semantics are not 'denotational' in the sense of Scott/ Strachey/Milner because the meaning of the procedure call construct is not defined in terms of the meanings of its components; they are thus partly operational in nature.

Milner explicitly mentions compositionality as basic principle (MILNER 1975, p.167):

If we accept that any abstract semantics should give a way of composing the meanings of parts into the meaning of the whole [..].

As motivation for this approach, he gives a very practical argument (ibid. p.158):

The designer of a computing system should be able to think of his system as a composite of behaviours, in order that he may factor his design problem into smaller problems [..].

Mazurkiewics mentions naturalness as a motivation for following the principle (MAZURKIEWICS 1975, p.75).

One of the most natural methods of assigning meanings to programs is to define the meaning of the whole program by the meanings of its constituents [..].

We observe that the principle arises in connection with semantics in many fields. In the philosophical literature the principle is almost always attributed to Frege, whereas in the fields of programming and natural language semantics this is not the case. Authors in these fields give a practical motivation for obeying the principle: one whishes to deal with an infinity of possibilities in some reasonable, practical, understandable, and (therefore) finite way.

2. FREGE AND THE PRINCIPLE

2.1. Introduction

In the previous section we observed that several philosophers attribute the principle of compositionality to Frege. But it is not made clear what the relationship is of the principle to Frege, and especially on what grounds the principle is attributed to him.

In his standard work on Frege, Dummett devotes a chapter to 'Some theses of Frege on sense and reference'. The first thesis he considers is (DUMMETT 1973, p.152):

> *The sense of a complex is compounded out of the senses of the constituents.*

Since sense is about the same as meaning (this will be explained later), the thesis expresses the principle of compositionality of meaning. Unfortunately, Dummett does not relate this thesis to statements in the work of Frege, so it remains unclear on what writings the claim is based that it is a thesis of Frege. The authors quoted in the previous section who attribute the principle to Frege, do not refer to his writings either.

In the previous section we met a remark by Popper stating that the principle is not explicit in Frege's work, but that it is certainly implicit. The connection with Frege is, according to Cresswell, even looser. He says (CRESSWELL 1973, p.75):

> *For historical reasons we call this Frege's principle. This name must not be taken to imply that the principle is explicitly stated in Frege.*

And in a footnote he adds to this:

> *The ascription to Frege is more a tribute to the general tenor of his views on the analysis of language.*

However, Creswell does not explain these remarks any further. So we have to conclude that the literature gives no decisive answer to the question what the relationship is of the principle to Frege. I will try to answer the question by considering Frege's publications and investigating what he explicitly says about this subject.

2.2. Grundlagen

The study of Frege's publications brings us to the point of terminology. Frege has introduced some notions associated with meaning, but his terminology is not the same in all his papers. Dummett says about 'Die Grundlagen der Arithmetik' (FREGE 1884) the following (DUMMETT 1973, p.193):

> *When Frege wrote 'Grundlagen', he had not yet formulated his distinction between sense and reference, and so it is quite possible that the words 'Bedeutung' and 'bedeuten', as they occur in the various statements* [..] *have the more general senses of 'meaning' and 'mean'* [..].

This means that in 'Grundlagen' we have to look for Frege's remarks concerning the 'Bedeutung' of parts. He is quite decided on the role of their Bedeutung (FREGE 1884, p.XXII):

> *Als Grundsätze habe ich in dieser Untersuchung folgende festgehalten:* [..] *nach der Bedeutung der Wörter muss in Satzzusammenhänge, nicht in ihrer Vereinzelung gefragt werden* [..]

He also says (FREGE 1884, p.73):

> *Nur im Zusammenhange eines Satzes bedeuten die Wörter etwas.*

Remarks like these ones are repeated, heavily underlined, several times in 'Grundlagen' (e.g. on p.71 and p.116). The idea expressed by them is sometimes called the principle of contextuality. Contextuality seems to be in conflict with compositionality for the following reason. The principle of compositionality requires that words in isolation have a meaning, since otherwise there is nothing from which the meaning of a compound expression can be built. A principle of contextuality would deny that words in isolation have a meaning.

Dummett discusses the remarks from 'Grundlagen', and he provides an interpretation in which they are not in conflict with compositionality (DUMMETT 1973, pp.192-196). A summary of his interpretation is as follows. The statements express that it has no significance to consider first the meaning of a word in isolation, and next some unrelated other question. Speaking about the meaning of a word makes only significance as preparation for considering the meaning of a sentence. The meaning of a word is determined by the role it plays in the meaning of the sentence.

Following Dummett's interpretation, the remarks from Grundlagen have not to be considered as being in conflict with the principle of compositionality. It is quite well possible to build the meaning of a sentence from the meanings of its parts, and to base at the same time the judgements about the meanings of these parts on the role they play in the sentences in which they may occur. As a matter of fact, this approach is often followed (for instance in the field of Montague grammar). In this way a bridge is laid between compositionality and contextuality. But even with his interpretation, the statements formulated in Grundlagen cannot be considered as propagating compositionality: nothing is said about building meanings of sentences from meanings of words.

Dummett's interpretation weakens the statements from 'Grundlagen' considerably. Unfortunately, Dummett hardly explains on which grounds he thinks that his interpretation coincides with Frege's intentions when writing 'Grundlagen'. He provides, for instance, no references to writings of Frege. I tried to do so, but did not find passages supporting Dummett's opinion. Dummett makes the remark that statements like the ones from 'Grundlagen' make no subsequent appearence in Frege's works. This is probably correct with respect to Frege's published works, but I found some statements which are close to those 'Grundlagen' in Frege's correspondence and in his posthumous writings. They do not express the whole context principle, but repeat the relevant aspect: that expressions outside the context of a sentence have no meaning. In a letter to E.V. Huntington, probably dating from 1902, Frege says the following (GABRIEL 1976, p.90).

> *Solche Zeichenverbindungen wie* "a+b", "f(a,b)" *bedeuten also nichts, und haben für sich allein keinen Sinn* [..]

In 'Einleitung in die Logik', dating from 1906, he says (HERMES 1969, p.204):

> *Durch Zerlegung der singulären Gedanken erhält man Bestandteile der abgeschlossenen und der ungesättigten Art, die freilich abgesondert nicht vorkommen.*

In an earlier paper (from 1880), called 'Booles rechnende Logik und die Begriffsschrift', he compares the situation with the behaviour of atoms (HERMES 1969, p.19).

> *Ich möchte dies mit dem Verhalten der Atome vergleichen, von denen man annimmt, dass nie eins allein vorkommt, sondern nur in einer Verbindung mit andern, die es nur verlässt, um sofort in eine andere einzugehen.*

The formulation of the statements from 'Grundlagen' is evidently in conflict with the principle of compositionality. From our investigations it appears that related remarks occur in other writings of Frege. This shows that the formulation used in 'Grundlagen' is not just an accidental, and maybe unfelicitous expression of his thoughts. For this reason, and for the lack of evidence for Dummett's interpretation, I am not convinced that Frege's clear statements have to be understood in a weakened way. I think that they should be understood as they are formulated. Therefore I conclude that in the days of 'Grundlagen' Frege probably would have rejected the principle of compositionality, and, in any case, the formulation we use.

2.3. Sinn und Bedeutung

In 'Ueber Sinn und Bedeutung' (FREGE 1892) the notions 'Sinn' and 'Bedeutung' are introduced. Frege uses these two already existing German words to name two notions he wished to discriminate. The subtle differences in the original meaning of these two words do not cover the different use Frege makes of them. For instance, it is very difficult to account for their differences in meaning in a translation. Frege himself has been confronted with these problems as appears from a letter to Peano (GABRIEL 1976, p.196):

> [..] *Sie* [..] *sagen, dass zwei deutschen Wörtern, die ich verschieden gebrauche, dasselbe italienische nach den Wörterbüchern entspreche. Am nächsten scheint mir dem Worte 'Sinn' das italienische 'senso' und dem Worte 'Bedeutung' das italienische 'significazione' zu kommen.*

Concerning the terminology DUMMETT (1973, p.84) gives the following information. The term 'Bedeutung' has come to be conventionally translated as 'reference'. Since 'Bedeutung' is simply the German word for 'meaning', one cannot render 'Bedeutung' as it occurs in Frege by 'meaning', without a special warning. The word 'reference' does not belie Frege's intention, though it gives it a much more explicit expression. Concerning 'Sinn', which is always translated 'sense', Dummett says that to the sense of a word or expression only those features of meaning belong which are relevant to the truth-value of some sentence in which it may occur. Differences in meaning which are not relevant in this way, are relegated by Frege to the 'tone' of the word or expression. In this way Dummett has given an indication what Frege intended with sense. It is not possible to be more precise about the meaning of 'Sinn'. As van Heyenoort says (Van HEYENOORT 1977, p.93):

> *As for the 'Sinn' Frege gives examples, but never presents a precise definition.*

And Thiel states (THIEL 1965, p.165):

> *What Frege understood as the 'sense' of an expression is a problem that is so difficult that one generally weakens it to the question of when in Frege's semantics two expressions are identical in sense (synonymous).*

I will not try to give a definition; it suffices for our purposes to conclude that the notion 'Sinn' is very close to the notion 'meaning'. Therefore we have to investigate Frege's publications after 1892 to see what he says about the compositionality of Sinn. What he says about compositionality of 'Bedeutung' is a different story (as illustration: he explicitly rejected that in 1919 (HERMES 1969, p.275), but this is not the case for compositionality of 'Sinn', as will appear in the sequel).

In 'Ueber Sinn und Bedeutung', I found one remark concerning the relation between the senses of parts and the sense of the whole sentence. Frege discusses the question whether a sentence has a reference, and, as an example, he considers the sentence *Odysseus wurde tief schlafend in Ithaka ans Land gesetzt.* Frege says that if someone considers this sentence as true or false, he assigns the name *Odysseus* a reference (Bedeutung). Next he says (FREGE 1892, p.33).

> *Nun wäre aber das Vordringen bis zur Bedeutung des Namens überflüssig: man könnte sich mit dem Sinne begnügen, wenn man beim Gedanken stehenbleiben wollte. Käme es nur auf den Sinn des Satzes, den Gedanken, an, so wäre es unnötig sich um die Bedeutung eines Satzteils zu kümmern; für den Sinn des Satzes kann ja nur der Sinn, nicht die Bedeutung dieses Teils in Betracht kommen.*

So Frege states that there is a connection between the sense of the whole sentence, and the senses of the parts. He does, however, not say anything about a compositional way of building the sense of the sentence. More in particular, the quotation is neither in conflict with the compositionality principle, nor with the statements from 'Grundlagen'. Therefore I agree with BARTSCH (1978), who says that, in 'Ueber Sinn und Bedeutung', Frege does not speak, as is often supposed, about the contribution of the senses of parts to the senses of the compound expression.

2.4. The principle

Up till now we have not found any statement expressing the principle of compositionality. But there are such fragments. The most impressive one is from 'Logik in der Mathematik', an unpublished manuscript from 1914 (HERMES 1969, p.243).

> *Die Leistungen der Sprache sind wunderbar. Mittels weniger Laute und Lautverbindungen ist sie imstande, ungeheuer viele Gedanken auszudrücken, und zwar auch solche, die noch nie vorher von einem Menschen gefasst und ausgedrückt worden sind. Wodurch werden diese Leistungen möglich? Dadurch, dass die Gedanken aus Gedankenbausteinen aufgebaut werden. Und diese Bausteine entsprechen Lautgruppen, aus denen der Satz aufgebaut wird, der den Gedanken ausdrückt, sodass dem Aufbau des Satzes aus Satzteilen der Aufbau des Gedankens aus Gedankenteilen entspricht. Und den Gedankenteil kann man den Sinn des entsprechendes Satzteils nennen, so wie man den Gedanken als Sinn des Satzes aufassen wird.*

This fragment expresses the compositionality principle. However, the fragment is not presented as a fragment expressing a basic principle. It is used as argument in a discussion, and does not get any special attention.

The quotation from 'Logik in der Mathematik', presented above, is considered very remarkable by the editors of Frege's posthumous works. They have added the following footnote in which they call attention to other statements of Frege which seem to conflict with the quotation in consideration (HERMES 1969, p.243)

> *An anderen Stellen schränkt Frege diesen Gedanken-Atomismus allerdings in dem Sinne ein, dass man sich die Gedanketeile nicht als van den Gedanken, in denen sie vorkommen, unabhängige Bausteine vorstellen dürfe.*

They give two references to such statements in Frege's posthumous writings (i.e. the book they are editors of). One is from 'Booles rechnende Logik..' (1880), the other from 'Einleitung in die Logik' (1906). I have quoted these fragments in the discussion of 'Grundlagen'. In this way the editors suggest that the fragment from 'Logik in der Mathematik' is a slip of the pen, and a rather incomplete formulation of Frege's opinion concerning these matters.

The fragment under discussion does, however, not stand on its own. Almost the same fragment can be found in 'Gedankenfüge' (FREGE 1923). I present the fragment here in its English translation from 'Compound thoughts' by Geach and Stoothoff (p.55).

> *It is astonishing what language can do. With a few syllables it can express an incalculable number of thoughts, so that even a thought grasped by a terrestrial being for the very first time can be put into a form of words which will be understood by someone to whom the thought is entirely new. This would be impossible, were we not able to distinguish parts in the thought corresponding to the parts of a sentence, so that the structure of the sentence serves as an image of the structure of the thought.*

Moreover, in a letter to Jourdain, written about 1914, Frege says (GABRIEL 1976, p.127):

> *Die Möglichkeit für uns, Sätze zu verstehen, die wir noch nie gehört haben, beruht offenbar darauf, dass wir den Sinn eines Satzes aufbauen aus Teilen, die den Wörtern entsprechen.*

It is a remarkable fact that all quotations propagating compositionality are written after 1910: 'Gedankenfüge' (1923), 'Logik in der Mathematik' (1914), letter to Jourdain (1914). I have not succeeded in finding such quotations in earlier papers. But the statements which seem to conflict with compositionality are from an earlier period: 'Booles rechnende Logik ....'(1880), 'Grundlagen' (1884), letter to Huntington (1902), 'Einleitung in die Logik' (1906). This shows that, say after 1910, Frege has written about these matters in a completely different way than before. From this I conclude that his opinion concerning these matters changed. On the other hand, Frege never put forward the idea of compositionality as a principle.

It was rather an argument, although an important one, in his discussions. I would therefore not conclude to a break in his thoughts; rather it seems me to be a shift in conception concerning a detail.

In the light of this change, the following information appears relevant. In 1902 Frege received a letter from Russell in which the discovery was mentioned of the famous contradiction in naive set theory, and, in particular, in the theory of classes in Frege's 'Grundgesetze'. About the influence of this discovery on Frege, Dummett says the following (DUMMETT 1973, p.657):

> *It thus seems highly probable that Frege came quickly to regard his whole programme of deriving arithmetic from logic as having failed. Such a supposition is not only probable in itself: it is in complete harmony with what we know of his subsequent carreer. The fourth period of his life may be regarded as running from 1905 to 1913, and it was almost entirely unproductive.*

For this reason I consider it as very likely that in this period Frege was not concerned with issues related to compositionality. Then it is understandable that after this period he writes in a different way about the detail of compositionality (recall that it never was a principle, but just an argument).

2.5. Conclusion

My conclusions are as follows. Before 1910, and in any case especially in the years when he wrote his most important and influential works, Frege would probably have rejected the compositionality principle, in any case the formulation we use nowadays. After 1910 his opinion appears to have changed, and he would probably have accepted the principle, in any case the basic idea expressed in it. However, Frege never put forward such an idea as a basic principle, it is rather an argument in his discussions. Therefore, calling the compositionality principle 'Frege's principle' is above all, honouring his contributions to the study of semantics. But it is also an expression of his final opinion on these matters.

3. TOWARDS A FORMALIZATION

In this section, I will give the motivation for a formalized version of the compositionality principle. It is *not* my purpose to formalize what Frege or other authors might have intended to say when uttering something like the principle. I rather take the principle in the given formulation as a starting point and proceed along the following line; the (formalized version of) the principle should have as much content as possible. This means that the principle should make it possible to derive interesting consequences about those grammars which are in accordance with the principle, and at the same time it should be sufficiently abstract and universal to be applicable to a wide variety of languages. From the formalization it must be possible to obtain necessary and sufficient conditions for a grammar to be in agreement with the compositionality principle.

Consider a language L which is to be interpreted in some domain D of meanings. The kind of objects D consists of depends on the language under consideration, and the use one wishes to make of the semantics. In this section such aspects are left unspecified. Defining the semantics of a language consists in defining a suitable relation between expressions in L and semantic objects in D. Then the compositionality principle says something about the way in which this relation between L and D has to be defined properly.

In the formulation of the principle given in section 1, we encounter the phrase 'its parts'. Clearly we should not allow the expressions of L to be split in some random way. In the light of the standard priority conventions, the expression $y+8$ is not to be considered as a *part* of the expression $7.y + 8.x$; so the meaning of $7.y + 8.x$ does not have to be built up from the meaning of $y + 8$. It would also be pointless to try to build the meaning of some compound expression directly from the meanings of its atomic symbols (the terminal symbols of the alphabet used to represent the language). Since distinct expressions consist of distinct strings of symbols, there is always some dependence of the meanings of the basic symbols. Consequently such an interpretation would trivialize the principle. Another trivialization results by taking all expressions of the language to be 'basic', and interpreting them individually. The principle is interesting only in case the 'parts' are not trivial parts. Traditionally, the true decomposition of an expression into parts is described in the syntax for the language. Thus a language, the semantics of which is defined in accordance with the principle, should have a syntax which clearly expresses what the parts of the compound expressions are.

Let the language L, together with the set of expressions we wish to consider as parts, be denoted by $\underline{E}$. In order to give the principle a nontrivial content, we assume that the syntax of the language consists of rules of the following form:

If one has expressions $E_1,\ldots,E_n$ *then one can build the compound expression* $S_j(E_1,\ldots,E_n)$.

Here S_j is some operation on expressions, and $S_j(E_1,\ldots,E_n)$ denotes the result of application of S_j to the arguments $E_1,\ldots,E_n$. If the rules have the above format, we define the notion 'parts of' as follows.

If expression E *is built by a rule* S_j *from arguments* $E_1,\ldots,E_n$, *then the parts of* E *are the expressions* $E_1,\ldots,E_n$.

It often is the case that a rule does not apply to all expressions in $\underline{E}$, and that certain groups of expressions behave the same in this respect. Therefore the set of expressions is divided into subsets. The names of these subsets are called *types* or *sorts* in logic, *categories* in linguistics, and *types* or *modes* in programming languages. Often the name of a set and the set itself are identified and I will follow this practice. Instead of speaking about elements of the subset of a certain type, I will speak about the elements of a certain type, etc. The use of names for subsets allows us to specify in each rule from which category its arguments have to be taken, and to which category the resulting expression belongs. Thus, a *syntactic rule* S_j has the following form:

If one has expressions $E_1,\ldots,E_n$ *of the categories* $C_1,\ldots,C_n$ *respectively, then one can form the expression* $S_j(E_1,\ldots,E_n)$ *of category* C_{n+1}.

An *equivalent formulation* is:

Rule S_j *is a function from* $C_1\times\ldots\times C_n$ *to* C_{n+1};
i.e. $S_j\colon C_1 \times \ldots \times C_n \rightarrow C_{n+1}$.

Suppose that a certain rule S_i is defined as follows:

$$S_i\colon C_1 \times C_2 \rightarrow C_3, \quad \text{where } S_i(E_1,E_2) = E_1E_2.$$

This means that S_i concatenates its arguments. Then our interpretation of the principle says that the meaning of E_1E_2 has to be built up from the meanings of E_1 and E_2. A case like this constitutes the most elementary version of the principle. A compound expression is divided into real subexpressions, and the meaning of the compound expression is built up from the meanings of these subexpressions. In such a case the formalization coincides with the simplest, most intuitive conception of the principle: parts are visible as parts. But in some situations one might wish to consider as part an expression which is not visible as a part. We will meet several examples in later chapters. One example is the phenomenon of discontinuous constituents. The phrase *take away* is not a subphrase of *take the apple away*; it is not visible as a part. Nevertheless, one might here wish to consider it as a unit which contributes to the meaning of *take the apple away*, i.e. as a part in the sense of the principle. The above definition of 'part' gives the possibility to do so. If the phrase *take the apple away* is produced by means of a rule which takes as arguments the phrases *the apple* and *take away*, then *take away* is indeed a part in the sense of the definition. The definition generalizes the principle for rules which are not just a concatenation operation, and consequently the parts need not be visible in the expression itself.

There are no restrictions on the possible effect of the rules S_j. They may concatenate, insert, permute, delete, or alter (sub)expressions of their arguments in an arbitrary way. A rule may even introduce symbols which do not occur in its arguments. Such symbols are called *syncategorematic symbols*. These are not considered as parts of the resulting expression in the sense of the principle, and therefore they do not contribute a meaning from which the meaning of the compound can be formed. I will assume in general that the rules are total (i.e. they are defined for all expressions of the required categories). In chapter 6 partial rules will be discussed.

The abstraction just illustrated implies that we have lost the most intuitive conception of the principle. But it is not unlikely that several authors who mention Frege's principle only have the most intuitive version in mind. In order to avoid confusion, I will call the more abstract version not 'Frege's principle', but 'the compositionality principle'. As for the simple rules, where only concatenation is used, our interpretation of the principle coincides with the most intuitive interpretation. In more complex cases, where it might not be intuitively clear what the parts are, our interpretation can be applied as well. If one wishes to stick to the most intuitive interpretation of the principle, one must use only concatenation rules. In that way one would restrict considerably the applicability of grammars satisfying the framework (see chapter 2, section 5).

So far we have not considered the possibility of ambiguities. It is not excluded that some expression E can be obtained both as $E = S_i(E_1,\dots,E_n)$ and as $E = S_j(E'_1,\dots,E'_m)$. In practice such ambiguities frequently arise. In a programming language (e.g. in Algol 68), the procedure identifier *random* can be used to denote the process of randomly selecting a number, as well as to denote the number thus obtained. The information needed to decide which interpretation is intended, is present in the production tree of the program, where the expression *random* is either of type 'real' or not. In natural languages ambiguities arise even among expressions of the same category.

Consider for instance the sentence *John runs or walks and talks.* Its meaning depends on whether *talks* is combined with *walks*, or with *runs or walks*. Also here, the information needed to solve the ambiguity is hidden in the production tree. In the light of such ambiguities, we cannot speak in general of the meaning of some expression, but only of its meaning with respect to a certain derivational history. If we want to apply the compositionality principle to some language with ambiguities, we should not apply it to the language itself, but to the corresponding language of derivational histories.

In computer science it is generally accepted that the derivational histories form the real input for the semantical interpretation. SCHWARTZ (1972, p.2) states:

> *We have sufficient confidence in our understanding of syntactic analysis to be willing to make the outcome of syntactic analysis, namely the syntax tree representation of the program, into a standard starting point for our thinking on program semantics. Therefore we may take the semantic problem to be that of associating a value* [..] *with each abstract program, i.e. parse tree.*

In the field of semantics of natural languages, it is also common practice not to take the expressions of the language themselves as input to the semantical interpretation, but structured versions of them. KATZ & FODOR (1963, p.503) write:

> *Fig. 6 shows the input to a semantic theory to be a sentence* S *together with a structural description consisting of the* n *derivations of* S, $d_1, d_2, \ldots, d_n$, *one for each of the* n *ways that* S *is grammatically ambiguous.*

The book of Katz and Fodor is one of the early publications about the position of semantics in transformational grammars. There has been a lot of discussion in that field concerning the part of the derivational history which may actually be used for the semantic interpretation. In the so-called 'standard theory' only a small part of the information is used: the 'deep structure'. In the 'extended standard theory', one also uses the information which 'transformations' are applied, and what the final outcome, the 'surface structure', is. In the most recent proposals, the view on syntax and its relation with semantics is rather different.

As a matter of fact, neither Schwartz, nor Katz and Fodor use the same (semantic) framework we have. They are quoted here to illustrate that the idea of using information from the derivational history as input to the semantic component is not unusual.

Let us now turn to the phrase 'composed from the meanings of its parts'. Consider again a rule S_i which allows us to form the compound expression $S_i(E_1,\ldots,E_n)$ from the expressions $E_1,\ldots,E_n$. Assume moreover that the meanings of the E_k are the semantical objects D_k. According to the principle, the information we are allowed to use for building the meaning of the compound expression consists in the meanings of the parts of the expression and the information which rule was applied. As usual, 'rule' is intended to take into account the order of its arguments. So the meaning of a compound expression is determined by an n-tuple of meanings (of its parts) and the information of the identity of the rule. This is in fact the only information which may be used. If one would be allowed to use other information (e.g. syntactic information concerning the parts), the principle would not express the whole truth, and not provide a sufficient condition. Thus the principle would tend to become a hollow phrase.

As argued for above, I interpret the compositionality principle as stating that the meaning of a compound expression is determined *completely* by the meanings of its parts and the information which syntactic rule is used. This means that for each syntactic rule there is a function on meanings which yields the meaning of a compound expression when it is applied to the meanings of the parts of that expression. So for each syntactic rule there is a corresponding semantic operation. Such a correspondence is not unusual; it can be found everywhere in mathematics and computer science. If one encounters for instance a definition of the style 'the function f*g is defined by performing the following calculations using f and g...', then one sees in fact a syntactic and a semantic rule. The syntactic rule introduces the operator * between functions and states that the result is again a function, whereas the semantic rule tells us how the function should be evaluated

If we use the freedom allowed by the principle at most, we may associate with each syntactic rule S_i a distinct semantic operation T_i. So the most general description of the situation is as follows. The meaning of an expression formed by application of S_i to $(E_1,\ldots,E_n)$ can be obtained by application of operator T_i to $(D_1,\ldots,D_n)$, where D_j is the meaning of E_j. These semantic operators T_i may be partially defined functions on the set $\underline{D}$ of meanings, since T_i has to be defined only for those tuples from $\underline{D}$ which may arise as argument of T_i. These are those tuples which can arise as meanings of arguments of the syntactic rule S_i which corresponds with T_i. In this way the set $\underline{E}$ leaves a trace in the set $\underline{D}$. The set of meanings of the expressions of some category forms a subset of $\underline{D}$ which becomes the set of possible arguments for some semantic rule. Thus the domain $\underline{D}$ obtains a structure which is closely related to the structure of the syntactic domain

E. Our *formalization* of the compositionality principle may *at this stage* be summarized as follows.

> *Let* $S_i: C_1 \times\ldots\times C_n \to C_{n+1}$ *be a syntactic rule, and* $M: \underline{E} \to \underline{D}$ *be a function which assigns a meaning to an expression with given derivational history. Then there is a function* $T_i: M(C_1) \times\ldots\times M(C_n) \to M(C_{n+1})$ *such that* $M(S_i(E_1,\ldots,E_n))=T_i(M(E_1),\ldots,M(E_n))$.

In the process of formalizing the principle of compositionality we have now obtained a special framework. The form of the syntactic rules and the use of sorts give the syntax the structure of, what is called, a 'many-sorted algebra'. The correspondence between syntax and semantics implicates that the semantic domain is a many-sorted algebra of the same kind as the syntactic algebra. The meaning assignment is not based upon the syntactic algebra itself, but on the associated algebra of derivational histories. The principle of compositionality requires that meaning assignment is a homomorphism from that algebra to the semantic algebra. Note that the principle, which is formulated as a principle for semantics, has important consequences not only for the semantics, but also for the syntax.

The approach described here, is closely related to the framework developed by the logician Richard Montague for the treatment of syntax and semantics of natural languages (MONTAGUE 1970b). It is also closely related to the approach propagated by the group called 'Adj' for the treatment of syntax and semantics of programming languages (ADJ 1977, 1979). Consequently frameworks related to the one described here can be found in the publications of authors following Adj (for references see ADJ 1979), or following Montague (for references see DOWTY, WALL & PETERS 1981, or the present book). The conclusion that the principle of compositionality requires an algebraic approach is also given by MAZURKIEWICS (1975) and MILNER (1975), without, however, developing some framework. The observation that there is a close relationship between the frameworks of Adj and Montague, was independently made by MARKUSZ & SZOTS (1981), ANDREKA & SAIN (1981), and Van EMDE BOAS & JANSSEN (1979).

4. AN ALGEBRAIC FRAMEWORK

In this section I will develop the framework sketched in section 3, and arguments concerning the practical use of the framework will influence this further development. The mathematical theory of the framework will be investigated in chapter 2.

The central notions in our formalization of the principle of compositionality are 'many-sorted algebra' and 'homomorphism'. An algebra consists of some set (the elements of the algebra), and a set of operations defined on those elements. A many-sorted algebra is a generalization of this. It consists of a non-empty set S of sorts (types, modes, or categories), for each sort $s \in S$ a set A_s of elements of that sort (A_s is called the carrier of sort s), and a collection $(F_\gamma)_{\gamma\in\Gamma}$ of operations which are mappings from cartesian products of specified carriers to a specified carrier. So in order to determine a many-sorted algebra, one has to determine a 'sorted' family of sets and a collection of operators. This should explain the following definition.

4.1. <u>DEFINITION</u>. A *many-sorted algebra* A is a pair $<(A_s)_{s\in S},\underline{F}>$, where
1. S is a non-empty set, its elements are called the sorts of A.
2. A_s is a set (for each $s\in S$), the carrier of sort s.
3. $\underline{F}$ is a collection of operators defined on certain n-tuples of sets A_s, where $n>0$.

4.1. END.

Structures of this kind have been defined by several authors, using different names; the name 'many-sorted algebra' is borrowed from ADJ(1977). Notice that in the above definition there are hardly any restrictions on the sets and operators. The carriers may be non-disjunct, the operators may perform any action, and the sets involved (except for S) may be empty.

In order to illustrate the notion 'many-sorted algebra', I will present three examples in an informal way. I assume that these examples are familiar, and I will, therefore, not describe them in detail. The main interest of these examples is that they illustrate the notion of a many-sorted algebra. The examples are of a divergent nature, thus illustrating the generality of this notion.

4.2. <u>EXAMPLE</u>: *Real numbers*

Let us consider the set of real numbers as consisting of two sorts. *Neg* and *Pos*. The carrier R_{Neg} of sort Neg consists of the negative real numbers, the carrier R_{Pos} of sort Pos of the positive real numbers, zero included. An example of an operation is *sqrt*: $R_{Pos} \to R_{Pos}$, where *sqrt* yields the square root of a positive number. For R_{neg} there is no corresponding operation. Since we consider (in this example) the real numbers as a two-sorted algebra, there are two operations for squaring a number. One for squaring a positive number (*sqpos:* $R_{Pos} \to R_{Pos}$) and one for squaring a negative number (*sqneg:* $R_{Neg} \rightarrow R_{Pos}$). Since these two operations are closely related, we may use the same symbol for both operations: $(\ )^2$.

4.3. EXAMPLE: *Monadic Predicate Logic*

Sorts are *Atom, Pred* and *Form*. The carrier A_{Atom} of sort Atom consists of the symbols $a_1, a_2, \ldots$ and the carrier A_{Pred} of the sort Pred consists of the predicate letters $P_1, P_2, \ldots$. The carrier A_{Form} consists of formulas like $P_1(a_1)$, $\neg P_1(a_1)$, and $P_1(a_2) \vee P_2(a_3)$. Two examples of operators are as follows.

1. The operation *Apl*: $A_{Pred} \times A_{Atom} \rightarrow A_{Form}$. Apl assigns to predicate P and atom a the formula where P is applied to a; viz. $P(a)$.
2. The operation *Disj*: $A_{Form} \times A_{Form} \rightarrow A_{Form}$. Disj assigns to two formulas ϕ and ψ their disjunction $\phi \vee \psi$.

Notice that (in the present algebraization) the brackets (,); and the disjunction symbol $\vee$ are syncategorematic symbols.

4.4. EXAMPLE: *English*

Examples of sorts are *Sentence, Verb phrase,* and *Noun phrase*. The carrier of sort Sentence consists of the analysis trees of English sentences, and the carriers of other sorts of trees for expressions of other sorts. An example of an operator is T_{Neg}: Sentence $\rightarrow$ Sentence. The operator T_{Neg} assigns to an analysis tree of an English sentence the analysis tree of the negated version of that sentence. An explicit and complete description of this algebra I cannot provide. This example is mentioned to illustrate that complex objects like trees can be elements of an algebra.
4.4. END.

As explained in the previous section, we do not assign meanings to the elements of the syntactic algebra itself, but to the derivational histories associated with that algebra. These derivational histories form an algebra: if expressions E_1 and E_2 can be combined to expression E_3, then the derivational histories of E_1 and E_2 can be combined to a derivational history of E_3. So the derivational histories constitute a (many-sorted) set in which certain operations are defined. Hence it is an algebra. The nature of the operations of this algebra will become evident when we consider below representations of derivational histories.

Suppose that a certain derivational history consists of first an application of operator S_1 to basic expressions E_1 and E_2, and of next an application of S_2 to E_3 and the result of the first step. A description like this of a derivational history is not suited to be used in practice (e.g. because of its verbosity). Therefore formal representations for derivational histories will be used (certain trees or certain mathematical expressions).

In Montague grammar one usually finds trees as representation of a derivational history. The history described above is represented in figure 1. Variants of such trees are used as well. Often the names of the rules are not mentioned (e.g. S_1,S_2), but their indices (viz. 1,2). Sometimes the rule, or its index is not mentioned, but the category of the resulting expression. Even the resulting expressions are sometimes left out, especially when the rules are concatenation rules (figure 2). The relation between the representations of derivational histories and the expressions of the languages is obvious. In figure 1 one has to take the expression labelling the root of the tree, and in figure 2 one has to perform the mentioned operations. (For this kind of trees, it usually amounts to a concatenation of the expressions mentioned at the leaves (i.e. end-nodes)).

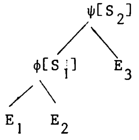


Figure 1. Representation of a derivational history

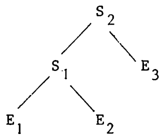


Figure 2. Another representation of the same derivational history

An alternative representation originates from the field of algebra. Derivational histories are represented by a compound expressions, consisting of basic expressions, symbols for the operators, and brackets. The derivational history from figure 1 is represented by the expression:

$S_2(S_1(E_1,E_2),E_3)$.

Such expressions are called terms. The algebra of terms corresponding with algebra A is called the term algebra T_A. From a term one obtaines an expression of the actual language by evaluating the term, i.e. by application of the operators (corresponding with the operator symbols) to the mentioned arguments. The sorts of the term algebra T_A are identical to the sorts of A, the operators are concatenation operators on terms. Note that all these different representations mathematically are equivalent.

MONTAGUE (1970b) introduced the name 'disambiguated language' for the algebra of derivational histories. The relation between the disambiguated language and the language under consideration (he calls it R) is completely arbitrary in his approach. The only information he provides is that it is a binary relation with domain included in the disambiguated language (MONTAGUE 1970b, p.226). From an algebraic viewpoint this arbitrariness is very unnatural, and therefore I restrict this relation in the way described above (evaluating the term, or taking the expression mentioned at the root). This is a restriction on the framework, but not on the class of languages that can be described by the framework (see chapter 2).

The tree representations are the most suggestive representations, and they are most suitable to show complex derivational histories. The term representations take less space and are suitable for simple histories and in theoretical discussions. In the first chapters I will mainly use terms, in later chapters trees. According to the framework we have to speak about the meaning of an expression relative to a derivational history. In practice one often is sloppy and speaks about the meaning of an expression (when the history is clear from the context, or when there is only one).

After this description of the notion of a many-sorted algebra, I will introduce the other central notion in our formalization of the principle of compositionality: the notion 'homomorphism'. It is a special kind of mapping between algebras, and therefore first mappings are introduced.

4.5. DEFINITION. By a *mapping* m from an algebra A to an algebra B is understood a mapping from the carriers of A to the carriers of B. Thus:

$$m: \bigcup_{s \in S_A} A_s \rightarrow \bigcup_{s \in S_B} B_s.$$

4.5. END.

A mapping is called a *homomorphism* if it respects the structures of the algebras involved. This is only possible if the two algebras have a similar structure. By this is understood that there is a one-one correspondence between the sorts in the one algebra and in the other algebra, and between the operators in the one algebra and in the other algebra. The latter means that if an operator is defined for certain sorts in the one algebra, then the corresponding operator is defined for the corresponding sorts in the other algebra. This should describe the essential aspects of the technical notion of 'similarity' of two algebras; a formal definition will be given in chapter 2. Then the definition of a homomorphism given below, will be adapted accordingly, and the slight differences with the definitions in the literature (Montague, Adj) will be discussed.

4.6. DEFINITION. Let $A = \langle (A_s)_{s \in S}, \underline{F} \rangle$ and $B = \langle (B_t)_{t \in T}, \underline{G} \rangle$ be similar algebras. A mapping h from A to B is called a *homomorphism* if the following two conditions are satisfied

1. h respects the sorts, i.e. $h(A_s) \subset B_t$, where t is the sort of B which corresponds to sort s of A.
2. h respects the operators, i.e. $h(F(a_1,\ldots,a_n)) = G(h(a_1),\ldots,h(a_n))$ where $G \in \underline{G}$ is the operator of B which corresponds to $F \in \underline{F}$.

4.6. END

Now that the notions of a many sorted algebra and of a homomorphism are introduced, I will present two detailed examples.

4.7. EXAMPLE: *Fragment of English.*

Syntactic Algebra

The syntactic algebra E consists of some English words and sentences

I. Sorts

S_E = {Sent,Subj,Verb}

II. Carriers

E_{Subj} = {*John, Mary, Bill*}
E_{Verb} = {*runs, talks*}
E_{Sent} = {*John runs, Mary runs, Bill runs, John talks, Mary talks, Bill talks*}

III. Operations

$C: E_{Subj} \times E_{Verb} \rightarrow E_{Sent}$
defined by $C(\alpha,\beta) = \alpha\beta$

So C(*John, runs*) is obtained by concatenating *John* and *runs*, thus yielding *John runs*.

Semantic Algebra

The semantic Algebra M consists of model-theoretic entities, such as truth values and functions.

I. Sorts

$S_M = \{e, t, <e,t>\}$

So there are three sorts: two sorts being simple symbols (e ~ entity, t ~ truthvalue), and the compound symbol <e,t> (function from e to t).

II. Carriers

$M_t = \{\underline{\text{true}}, \underline{\text{false}}\}$	The carrier M_t consists of two elements, the truth-values <u>true</u> and <u>false</u>.
$M_e = \{e_1, e_2, e_3\}$	The set M_e consists of three elements: e_1, e_2 and e_3.
$M_{<e,t>} = (M_t)^{M_e}$	The carrier $M_{<e,t>}$ consists of all functions from M_e to M_t. This set has 8 elements.

III. Operations

There is one operation in M: the operation F of function application.

$F: M_e \times M_{<e,t>} \rightarrow M_t$,

where $F(\alpha;\beta)$ is the result of application of β to argument α.

The algebras E and M are similar. The correspondence of sorts is Subj ~ e, Verb ~ <e,t>, Sent ~ t, and operation C corresponds to F. Although the algebras E and M are similar, they are not the same. For instance, the number of elements in E_{verb} differs from the number of elements in $E_{<e,t>}$.

There are a lot of homorphisms from T_E (the derivational histories in E), to M. An example is as follows.

Let h be defined by

h(*John*) = e_1, h(*Bill*) = e_2, h(*Mary*) = e_3

h(*runs*) is the function f_1 which has value <u>true</u> for all $e \in M_e$

h(*talks*) is the function f_2 which has value <u>false</u> for all $e \in M_e$

Furthermore we define h for the compound terms.

h(C(*John*,*runs*)) = h(C(*Mary*,*runs*)) = h(C(*Bill*,*runs*)) = <u>true</u>

h(C(*John*, *talks*)) = h(C(*Mary*, *talks*)) = h(C(*Bill*, *talks*)) = <u>false</u>

The function h, thus defined, is a homomorphism because

1. $h(T_{E,Subj}) \subset M_e$, $h(T_{E,Verb}) \subset M_{<e,t>}$, $h(T_{E,sent}) \subset M_E$
2. $h(C(\alpha,\beta)) = F(h(\alpha),h(\beta))$ for all subjects α and verbs β.

It is easy to define other homomorphisms from T_E to M. Notice that once h is defined for $T_{E,Subj}$ and for $T_{E,Verb}$, then there is no choice left for the definition of h for $T_{E,Sent}$ (provided that we want h to be a homomorphism).

4.8. EXAMPLE: *Number denotations*

Syntactic Algebra

The algebra *Den* of natural number denotations is defined as follows

I. Sorts

S_{Den} = {digit,num}

II. Carriers

D_{digit} = {*0,1,2,3,4,5,6,7,8,9*}

D_{num} = {*0,1,2,3,...10,11,...01,02,..010,..001,..007,.....*}

So D_{num} is the set of all number denotations, including denotations with leading zero's. Notice that $D_{digit} \subset D_{num}$.

III. Operators

There is one operation.

C: $D_{num} \times D_{digit} \rightarrow D_{num}$

where C is defined by $C(\alpha,\beta) = \alpha\beta$.

So C concatenates a number with a digit.

Semantic Algebra

The algebra *Nat* of natural numbers is defined as follows

I. Sorts

S_{Nat} = {d,n}.

II. Carriers

N_d consists of the natural numbers up to nine (zero and nine included)

N_n consists of all natural numbers.

III. Operations

There is one operation:

F: $N_n \times N_d \rightarrow N_n$

where F is defined as multiplication of the element from N_n by ten, followed by addition of the element from N_d.

A natural homomorphism h from T_{Den} to *Nat* is the mapping which associates with the derivational history of a digit or number denotation the corresponding number. Then h(C(*0,7*)) and h(*7*) are both mapped onto the number seven. That this h is a homomorphism follows from the fact that F describes the semantic effect of C, e.g. h(C(*2,7*)) = F(h(*2*),h(*7*)).

4.8. END

Syntax is an algebra, semantics is an algebra, and meaning assignment is a homomorphism; that is the aim of our enterprise. But much work has to be done in order to proceed in this way. Consider the two examples given above. The carriers were defined by specifying all their elements, the homomorphisms were defined by specifying the image of each element, and the operations in the semantic algebra were described by means of full English sentences. For larger, more complicated algebras this approach will be very impractical. Therefore a lot of technical tools will have to be introduced before we can deal with an interesting fragment of natural language or programming language. Consider again the first example (i.e. 4.7). The semantic operation corresponding to the concatenation of a *Subj* and a *Verb* was described as the application of the function corresponding to the verb to the element corresponding to the subject. One would like to use standard notation from logic and write something like *Verb(Subj)*. Thus one is tempted to use some already known language in order to describe a semantic operation. This is precisely the method we will employ. If we wish to define the meaning of some fragment of a natural language, or of a programming language, we will not describe the semantic operations in the meta-language (for instance a mathematical dialect of English), but use some formal language, the meaning of which has already been defined somehow: we will use some formal or logical language. Thus the meaning of an expression is defined in two steps: by translating first, and next interpreting, see figure 3.

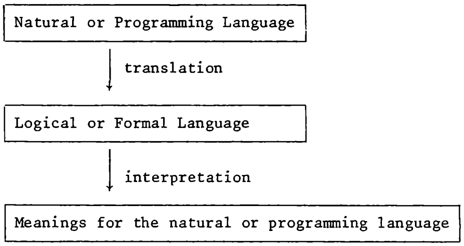

Natural or Programming Language
translation
Logical or Formal Language
interpretation
Meanings for the natural or programming language

Figure 3 Meaning assignment in two steps.

Figure 3 illustrates that the semantics of the fragment of English is defined in a process with two stages. But is this approach in accordance with our algebraic aim? Is the mapping *from* the term algebra corresponding with the syntax of English *to* the algebra of meanings indeed a homomorphism? The answer is that we have to obey certain restrictions, in order to be sure that the two-stage process indeed determines a homomorphism. The translation should be a homomorphism from the term algebra for English to the logical language and the interpretation of the logical language should be a homomorphism as well. Then, as is expressed in the theorem below, the composition of these two mappings is a homomorphism.

4.9. <u>THEOREM</u>. *Let* A,B, *and* C *be similar algebras, and* $h: A \to B$ *and* $g: B \to C$ *homomorphisms. Define the mapping* $h\circ g: A \to C$ *by* $(h\circ g)(a) = g(h(a))$. *Then* $h\circ g$ *is a homomorphism.*

<u>PROOF</u>.
1. $(h\circ g)(A_s) \subset g(h(A_s)) \subset g(B_{s'}) \subset C_{s''}$, where s' and s" are the sorts in B and C corresponding with s.
2. Let G_γ, H_γ be the operators in B and C corresponding with F_γ. Then $(h\circ g)(F_\gamma(a_1,\ldots,a_n)) = g(h(F_\gamma(a_1,\ldots,a_n))) = g(G_\gamma(h(a_1),\ldots,h(a_n))) =$ $= H_\gamma(g(h(a_1),\ldots,g(h(a_n))) = H_\gamma((h\circ g)(a_1),\ldots,(h\circ g)(a_n))$

4.9. END.

The semantical language does not always contain basic operators which correspond to the operators in the syntax. In the example concerning natural number denotations there is no basic arithmetical operator which corresponds to the syntactic operation of concatenation with a digit. I described the semantic operator by means of the phrase 'multiplication of the element from N_n with ten; followed by an addition with the element of N_d'. One is tempted to indicate this operation not with this compound phrase, but with something like '10 × number + digit'. One wishes to use a compound expression from the language of arithmetic for the semantic operation which corresponds to the concatenation operation, i.e. to build new operations from old ones.

The situation I have just described, is the one which almost always arises in practice. One wishes to define the semantics of some language. The set of semantic objects has some 'natural' structure of its own, and a 'natural' semantical language which reflects this structure. So this 'natural' semantical language has not the same algebraic structure as the language for which we wish to describe the semantics. Therefore we use the semantical language (usually some kind of formal or logical language) to build a new algebra, called a derived algebra. We make new operations by forming compound expressions which correspond with the syntactic operations of the language for which we wish to describe the semantics. This situation is presented in figure 4; the closed arrows denote mappings, the dotted arrows indicate the construction of a new algebra by means of the introduction of new operations (built from old ones).

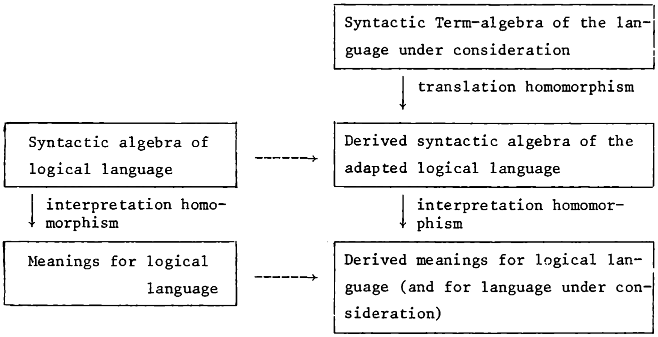

Syntactic Term-algebra of the language under consideration
translation homomorphism
Syntactic algebra of logical language
Derived syntactic algebra of the adapted logical language
interpretation homomorphism
interpretation homomorphism
Meanings for logical language
Derived meanings for logical language (and for language under consideration)

Figure 4. Meaning assignment using derived algebras

In this way, we have derived a new syntactic algebra from the syntactic algebra of the logical language. The syntactic algebra of which we wish to define the semantics is translated into this derived algebra. Now the question arises whether this approach is in accordance with our aim of defining some homomorphism from the syntactic algebra to the collection of meanings. The theorem that will be mentioned below guarantees that under certain conditions this is the case. The interpretation of the logical language has to be a homomorphism, and the method by which we obtain the derived algebra is restricted to the introduction of new operators by composition of old operators. Such operators are called polynomials; for a formal description see chapter 2. If these conditions are satisfied, then the interpretation homomorphism for the logical language is also an interpretation homomorphism of the derived algebra (when restricted to this algebra). Composition of this interpretation homomorphism with the translation homomorphism gives the desired homomorphism from the language under consideration to its meanings. The theorem is based upon MONTAGUE (1970b), for its proof see chapter 2.

4.10. THEOREM. *Let* A *and* B *be similar algebras and* h: A → B *a homomorphism onto* B. *Let* A' *be an algebra obtained from* A *by means of introduction of polynomially defined operators over* A.
Then there is a unique algebra B' *such that* h *is a homomorphism from* A' *onto* B'.
4.10. END

Finally I wish to make some remarks about the translation into some logical language. As I explained when introducing this intermediate step, it is used as a tool for defining the homomorphism from the syntactic algebra to the semantic one. If we would appreciate complicated definitions in the meta language, we might omit the level of a translation. It plays no essential role in the system, it is there for convenience only. If convenient, we may replace a translation by another translation which gets the same interpretation. We might even use another logical language. So in a Montague grammar there is nothing which deserves the name of *the* logical form of an expression. The obtained translation is just one representation of a semantical object, and might freely be interchanged with some other representation. KEENAN & FALTZ (1978), in criticizing the logical form obtained in a Montague grammar, criticize a notion which does not exist in Montague grammar.

5. MEANINGS

5.1. Introduction

In this section some consequences are considered of the requirement of a homomorphic mapping from the syntactic term algebra to the semantic algebra. These consequences are considered for three kinds of language: natural languages, programming languages and logical languages. It will appear that the requirement of associating a single meaning with each expression of the language helps us, in all three cases, to find a suitable formalization of the notion of meaning. Furthermore, an example will be considered of an approach where the requirement of a homomorphic relation between syntax and semantics is *not* obeyed.

5.2. Natural Language

Consider the phrase *the queen of Holland*, and assume that it is used to denote some person (and not an institution). Which person is denoted, depends on the moment of time one is speaking about. This information can usually be derived from the linguistic context in which the expression occurs. In

(1) *The queen of Holland is married to Prince Claus.*

Queen Beatrix is meant, since she is the present queen. But in

(2) *In 1910 the queen of Holland was married to Prince Hendrik.*

Queen Wilhelmina is meant, since she was the queen in the year mentioned. So one is tempted to say that the meaning of the phrase *the queen of Holland* varies with the time one is speaking about. Such an opinion is, however, not in accordance with our algebraic (compositional) framework. The approach which leads to a single meaning for the phrase under discussion is to incorporate the source of variation into the notion of meaning. In this way we arrive at the conception that the meaning of the phrase *the queen of Holland* is a function from moments of time to persons. For other expressions (and probably also for this one) there are more factors of influence (place of utterance, speaker,..). Such factors are called indices; a function with the indices as domain is called an intension. So the meaning of an expression is formalized by an intension: our framework leads to an intensional conception of meaning for natural language. For a more detailed discussion concerning this conception, see LEWIS 1970. A logical language for dealing with intensions is the language of 'intensional logic'. This language will be considered in detail in chapter 3.

5.3. Programming Language

Consider the expression $x+1$. This kind of expressions occurs in almost every programming language. It is used to denote some number. Which number is denoted depends on the internal situation in the computer at the moment of consideration. For instance, in case the internal situation of the computer associates with x the value seven, then $x+1$ denotes the number eight. So one is tempted to say that the meaning of $x+1$ varies. But this is not in accordance with the framework. As in example 1, the conflict is resolved by incorporating the source of variation into the notion of meaning. As the meaning of an expression like $x+1$ we take a function from computer states to numbers. On the basis of this conception a compositional treatment can be given of meanings of computer languages (See chapter 10). States of the computer can be considered as an example of an index, so also in this case we use an intensional approach to meaning. In the publications of Adj a related conception of the meaning of such expressions is given, although without calling it an intension (see e.g. ADJ 1977, 1979).

Interesting in the light of the present approach is a discussion in PRATT 1979. Pratt discusses two notions of meaning: a static notion (an expression obtains once and for all a meaning), and a dynamic notion (the meaning of an expression varies). He argues that (what he takes as) a static notion of meaning has no practical purpose because we frequently use expres-
pression obtains once and for all a meaning), and a dynamic notion (the
of time. Therefore he develops a special logic for the treatment of semantics of programming languages, called 'dynamic logic'. But on the basis of our framework, we have to take a 'static' notion of meaning. By means of intensions we can incorporate all dynamics into such a framework. Pratt's dynamic meanings might be considered as a non-static version of intensional logic.

5.4. <u>Predicate Logic</u>

It is probably not immediately clear how predicate logic fits into the algebraic framework. PRATT (1979,p.55) even says that 'there is no function F such that the meaning of $\forall xp$ can be specified with a constraint of the form $\mu(\forall xp) = F(\mu(p))$'. In our algebraic approach we have to provide for such a meaning function μ and operator F.

Let us consider the standard (Tarskian) way of interpreting logic. It roughly proceeds as follows. Let @ be a model and g be an @-assignment. The interpretation in @ of a formula ϕ with respect to g, denoted ϕ^g, is then recursively defined. One clause of this definition is as follows (here 1 denotes the truth value for truth).

$[\phi \wedge \psi]^g$ is 1, if ϕ^g is 1 and ψ^g is 1.

This suggest that the meaning of $\phi \wedge \psi$ is a truth value, which is obtained out of the truth values for ϕ and for ψ. Another clause of the standard way of interpretation is not compatible with this idea.

$[\exists x\phi(x)]^g$ is 1, if there is a $g' \underset{x}{\sim} g$ such that $[\phi(x)]^{g'}$ is 1.

(Here $g' \underset{x}{\sim} g$ means that g' is the same assignment as g except for the possible difference that $g'(x) \neq g(x)$).

This clause shows that the concept of meaning being a truth value is too simple for our algebraic framework. One cannot obtain the truth value of

$\exists x\phi(x)$ (for a certain value of g) out of the truth value of $\phi(x)$ (for the same g). If we wish to treat predicate logic in our framework, we have to find a more sophisticated notion of meaning for it.

Note that there is not a single truth value in the semantic domain which corresponds with $\phi(x)$. Its interpretation depends on the interpretation of x, and in general on the interpretation of the free variables in ϕ, and therefore on g. In analogy with the previous examples, we incorporate the variable assignment into the conception of meaning. The meaning of a formula is a function from variable assignments to truthvalues, namely the function which yields 1 for an assignment in case the expression is true for that assignment. With this conception, we can easily build the meaning of $\phi \wedge \psi$ out of the meaning of ϕ and of ψ: a function which yields 1 for an assignment iff both the meanings of ϕ and of ψ yield 1 for that assignment. The formulation becomes simpler by adopting a different view of the same situation. A function from assignments to truthvalues can be considered as the characteristic function of a set of assignments. Using this, we may formulate an alternative definition: the meaning of a formula is a set of variable assignments (namely those for which the formula gets the truth value 1). Let M denote the meaning assignment function. Then we have:

$M(\phi\wedge\psi) = M(\phi) \cap M(\psi)$.

For the other connectives there are related set theoretical operations. Thus this part of the semantic domain gets the structure of a Boolean algebra.

For quantified formulas we have the following formulation.

$M(\exists x\phi) = \{h \mid h \underset{x}{\sim} g \text{ and } g \in M(\phi)\}$.

Let C_x denote the semantical operation described at the right hand side of the = sign, i.e. C_x is the operation 'extend the set of assignments with all x variants'. The syntactic operation of writing $\exists x$ in front of a formula now has a semantic interpretation: namely apply C_x to the meaning of ϕ.

$M(\exists x\phi) = M(\exists x)(M(\phi)) = C_x M(\phi)$.

In this algebraization there are infinitely many operations which introduce the existential quantifier. One might wish to go one step further and produce $\exists x$ from $\exists$ and x. This would require that given the meaning of a variable (being a function from assignments to values) we are able to determine of which variable it is a meaning. This is not an attractive algebraic operation, and therefore this last step is not made. I conclude that we have obtained a compositional interpretation of predicate logic: a homomorphism to some semantic algebra. One might say that it shows how we have to look at the Tarskian interpretation of logic in order to give it a compositional perspective.

The view on the semantics of predicate logic presented here is not new. Some logic books are based on this approach in which the meaning of a quantified formula is a set of assignments (MONK 1976, p.196, KREISEL & KRIVINE 1976, p.17). The investigations on the algebraic structure of predicate logic constitute a special branch of logic: the theory of cylindric algebras. It requires a shift of terminology to see that the kinds of structures studied there is the same as those introduced here. An assignment can be considered as an infinite tuple of elements in the model: the first element of the tuple is the value for the first variable, etcetera. Thus an assignment can be considered as a point in an infinite dimensional space. So if ϕ holds for a set of assignments, then ϕ is interpreted as the set of corresponding points in this universe. The operator C_x applied to a point p causes that all points are added which differ from p only in their x-coordinate. Geometrically speaking, a single point extends to an infinite stick. If C_x is applied to a set consisting of a circle area, then this is extended to a cylinder. Because of this effect, C_x is called a cylindrification operator, and in particular, the x-th cylindrification. (see fig.5) The algebraic structure obtained in connection with predicate logic is called a cylindric set-algebra. These algebras and their connection with logic are studied in the theory of cylindric algebras (see HENKIN, MONK & TARSKI 1971).

The original motivation for studying cylindric algebras was a technical one. Cylindric algebras were introduced to make the application of the powerful tools of algebra possible in studying logics, as can be read in HENKIN, MONK & TARSKI (1971, p.1):

> *This theory* [..] *was originally designed to provide an apparatus for an algebraic study of first order, predicate logic.*

New in the above discussion was the motivation which led us towards cylindric algebras. In my opinion, the compositional approach gives rise to a more direct introduction to this field than the existing one. Moreover, on the basis of the approach given above, it is not too difficult to find algebras for other order logics, such as intensional logic.

It cannot be said that the theory of cylindric algebras itself is a flourishing branch of logic nowadays. But the use of algebra is widespread in model theory (i.e. the branch of logic which deals with interpretations). Often one uses the terminology and techniques from universal algebra, as is evidenced by the amount of universal algebra in 'Model theory' by CHANG &

KEISLER (1973), and by the amount of model theory in 'Universal algebra' by GRAETZER (1968). Results from one field are proven using methods from the other field in Van BENTHEM (1979b). Algebraic interpretations of several non-classical logics are given by RASIOWA (1974). Important results concerning modal logics are obtained, using algebraic techniques, by Blok (e.g. BLOK 1980).

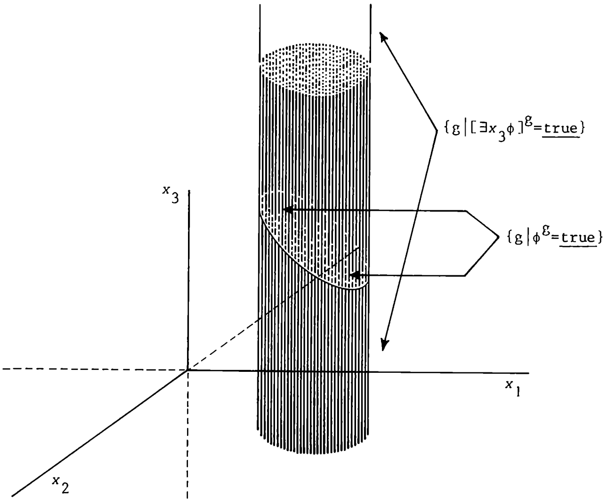

$\{g \mid [\exists x_3 \phi]^g = \underline{true}\}$
$\{g \mid \phi^g = \underline{true}\}$
x_3
x_1
x_2

Figure 5. A cylindrification

The present discussion should not be understood as claiming that the only legitimate way of studying (predicate) logic is by means of (cylindric) algebras. There are a lot of topics concerning logic that can be studied, and each has a natural viewpoint. For instance, if one is studying deduction systems, a syntactic point of view is the natural approach. One should take that view which is the best for one's current aims. What I claim is that, if one is studying semantics, then there has to be an algebraic interpretation existing in the background, and one should take care that this interpretation is not violated by what one is doing.

5.5. Strategy

In all three examples discussed above, we followed the strategy of first investigating what a meaning should do, and then defining such an entity as the formalized notion of meaning which does that and which satisfies the compositionality principle. In all examples such an entity was obtained by giving the notion of meaning a sufficient degree of abstraction. By proceeding in this way (first investigating, then defining) we follow the advice of LEWIS (1970,p.5)

In order to say what a meaning is, we may first ask what a meaning does and then find something that does that.

5.6. Substitutional Interpretation

Next I will discuss an approach to the semantics of predicate logic which is not compositional with respect to the interpretation of quantifiers. For the interpretation of $\exists x[\phi(x)]$ an alternative has been proposed which is called the 'substitutional interpretation'. It says:

$\exists x\ \varphi(x)$ is true iff there is some substitution a for x such that $\phi(a)$ is true.

Whether this definition is semantically equivalent to the Tarskian definition depends, of course, on whether the logical language contains a name for every element of the semantic domain or not. A definition like the above one can be found in two rather divergent branches of logic: in philosophical logic, and in proof theory.

In *philosphical logic* the substitutional interpretation has been put forward by R. Marcus (e.g. MARCUS 1962). Her motivation was of an ontological nature. Consider sentence (3).

(3) *Pegasus is a winged horse.*

According to standard logic, (4) is a consequence of (3), and Marcus accepts this consequence.

(4) $\exists x$(*x is a winged horse*).

She argues, however, that one might believe (3), without believing (5).

(5) *There exists at least one thing which is a winged horse.*

This opinion has as a consequence that the quantification used in (4) cannot be considered as an existential quantification in the ontological sense. The substitutional interpretation of quantifiers allows her to have (4) as a consequence of (3), without being forced to accept (5) as a consequence.

KRIPKE (1976) discusses this approach in a more formal way. As syntax for the logic he gives the traditional syntax: $\exists x\phi(x)$ is produced from $\phi(x)$ by placing $\exists x$ in front of it. According to such a grammar $\phi(a)$ certainly is not a part of $\exists x\phi(x)$. This means that the substitution interpretation is not a compositional interpretation (this was noticed by Tarski, as appears from a footnote in PARTEE (1973,p.74)).

In *proof theory* the substitutional interpretation is given e.g. in SCHUETTE 1977. In his syntax he constructs $\forall x\phi(x)$ from $\phi(a)$, where a is arbitrary. So the formula $\forall x\phi(x)$ is syntactically rather ambiguous: It has as many derivations as there are expressions of the form $\phi(a)$. Given one such production, it is impossible to define the interpretation of $\forall x\phi(x)$ on the basis of the interpretation of the formula $\phi(a)$ from which $\forall x\phi(x)$ was built in the parse under consideration. It may be the case that $\forall x\phi(x)$ is false, and $\phi(a)$ is true for some a, but false for another one. So we see that the truth value of $\forall x\phi(x)$ cannot depend on the truth value of $\phi(a)$ for any single a. Hence in this case the substitutional interpretation does not satisfy the compositionality principle.

If one wishes to define the semantics in a compositional way, and to follow at the same time the substitutional interpretation of quantifiers, then the syntax has to contain an infinitistic rule which says that *all* expressions of the form $\phi(a)$ are part of $\forall x\phi(x)$. Such an infinitistic rule has not been proposed by authors which follow the substitutional interpretation.

6. MOTIVATION

In this section I will give several arguments for accepting the compositionality principle and the formalization given for it. I will give three kinds of arguments. The first kind is very general and argues for working within some mathematically defined framework. The second kind of arguments lists benefits of working with the present framework, and is based upon the properties of the framework. The third kind concerns the principle itself. As a matter of fact, this entire book is intended as a support for the algebraic formalization of the compositionality principle, and many of the arguments will be worked out in the remainder of this book.

Regarding the *first* kind of *arguments*: it is very *useful* to work within some mathematically well defined framework. Such a standard framework gives rise to a language in which one can formulate observations, relations and generalizations. It is a point of departure for formulating extensions, restrictions and deviations. If one has no standard framework, then whenever one considers a new proposal, one has to start anew in obtaining intuitions concerning properties of the system, and to check whether old knowledge still holds. It is then difficult to see whether the proposals within some framework are in accordance with those in other frameworks, and whether they can be combined into a coherent treatment. If one wishes to design a computer program for Montague grammars, then one has to design for each proposed extension or variant a completely new program, unless all proposals fit into a single framework. This experience was my original motivation for the whole research presented in this book. But the final result is independent of this motivation: only at a few places programming considerations are mentioned (viz. here and in chapters 7 and 8).

The *second* kind of *arguments* is based upon the *quality* of the framework.

a) *Elegance*

The framework presented here is mathematically rather elegant. This is apparent especially from the fact that it is based upon two simple mathematical notions: many-sorted algebra and homomorphism. The important tool of a logical language is combined in an elegant way with these algebraic notions. One should, however, not confuse the notion of 'elegant' with 'elementary' or 'easy to understand'. That the system is elegant, is due to its abstractness, and this abstractness might be a source of difficulties in understanding the system. The insight obtained from the abstract view on the framework led to an answer to a question of PARTEE 1973 concerning restrictions on relative clause formation, see chapter 9 or JANSSEN 1981a. It also led to an application in a rather different direction by providing a semantics for Dik's functional grammar, see JANSSEN 1981b.

b) *Generality*

The framework can be applied to a wide variety of languages: natural, programming and logical languages. See chapter 3 for an application to logic, chapter 10 for an application to programming languages, and the other chapters of this book for applications to natural languages.

c) *Restrictiveness*

The framework gives rise to rather strong restrictions concerning the organization of syntax and semantics, and their mutual relation. The use of polynomial operators especially constitutes a concrete, practical restriction. For a discussion of several deviations from the present framework, see chapters 5 and 6.

d) *Comprehensibility*

The argument given by Milner (see section 1) for designers of computing systems can be generalized to: 'if someone describes the semantics of some language, he should be able to think of the description as a composite of descriptions, in order that he may factor a semantic problem into smaller problems'. And what is said here for the designer of a system, holds at least as much for someone trying to understand the system. This property of the system is employed in the presentation of the fragment in chapter 4.

e) *Power*

The recursive definitions used in the framework allow us to apply the technique of induction. Statements concerning structures and expressions can be proved by using induction to the complexity of the elements involved. Especially in chapters 2 and 3 this power is employed.

f) *Heuristic tool*

A most valuable argument in favor of the principle and its formalization is its benefit for the practice of describing semantics of languages. Examples of this benefit, however, would require a detailed knowledge of certain proposals. Therefore some quotations have to suffice.

ADJ 1979 (p.85) say about the algebraic approach:

> *The belief that the ideas presented here are key, comes from our experience over the last eight years in developing and applying these concepts.*

Furthermore they say (op.cit.p.88):

> *When one becomes familiar with such concepts (and the results concerning them) they provide a guide as to what one should look for, and as to how to formulate one's definitions and results.*

Van EMDE BOAS & JANSSEN 1979 (p.112) claim:

> *It will turn out that quite often some complicated description in a semantic treatment actually hides a deviation from the principle. Confronted with such a violation the principle sometimes suggests an alternative approach to the problematic situation which does obey the principle and solves the problem easier than thought to be possible. Such cases establish the value of the principle as a heuristic tool.*

Both papers contain a lot of evidence for their claims. I will present several examples supporting them: concerning programming languages in chapter 10, and concerning natural languages in the other chapters.

The *last* kind of *arguments* concerns the *principle itself*.

g) *No alternative*

An important argument in favor of the principle is that there is no competing principle. Authors not working in accordance with the principle do not, as far as I know, put forward an alternative general principle with a mathematical formalization. The principles one finds in the literature are language-specific, or specific for a certain theory of languages, but never principles concerning a framework.

h) *Widespread*

As demonstrated in section 1, the principle of compositionality is widespread in sciences dealing with semantics; it arises in philosophy, linguistics, logic and computer science.

i) *Psychology*

An argument sometimes put forward is that the principle reflects something of the way in which human beings understand natural language. The principle explains how it is possible that a human being, with his finite brain, can understand a potentially infinite set of sentences. Or to say it in Frege's words (as translated by Geach & Stoothof (FREGE 1923, p.35)):

> [..] *even a thought grasped by a terrestrial being for the first time can be put into a form of words which will be understood by someone to whom the thought is entirely new. This would be impossible, were we not able to distinguish parts in the thought corresponding to the parts of a sentence* [..].

The last two arguments I do not consider as very strong. As for argument h), I think that the principle is so popular because it is so vague. There are many undefined words in the formulation of the principle, so that everybody can find his own interpretation in it. As for argument i), we know so little about the process in the human brain associated with learning or understanding natural language, that arguments concerning psychological relevance are no more than speculations. I would not like to have the mathematical attractiveness of the framework disturbed by further speculations of this nature. The most valuable arguments are, in my opinion, those concerning the elegance and power of the framework, its heuristic value, and the lack of a mathematically well defined alternative. So I adhere to the principle for the technical qualities of its formalization.

An argument *not* found above is the *truth* of the principle: a statement like 'The semantics of English is compositional'. Such an argument would not be convincing since it is circular. In section 5, I gave examples which illustrated that the principle, and especially the requirement of similarity, may lead us to a certain conception of meaning. And in section 3 I gave a definition of the notion 'parts' which made it possible to have 'abstract parts'. So there is a large freedom: we may choose what the parts are of an expression, and what the meanings are of those parts. In such a situation it is not surprising that there is some choice which gives rise to a compositional treatment of the semantics. In the next chapter I will prove that it is possible within this framework to generate every recursively enumerable language, and to relate with every sentence any meaning we would like. If someone wishes to doubt the principle, this only seems possible if he has some judgements at forehand about what the parts of an expression are, and what their meanings are. In the light of the power and flexibility of the framework, it cannot be refuted by pointing out in some language a phenomenon which requires a non-compositional treatment. I expect that problematic cases can always be dealt with by means of another organization of the syntax, resulting in more abstract parts, or by means of a more abstract conception of meaning. The principle only has to be abandoned if it leads too often to unnecessarily complicated treatments.

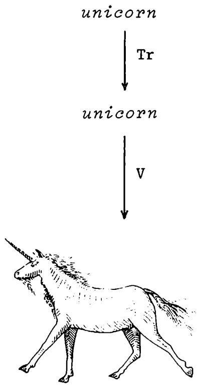
As appears from this discussion, the principle of compositionality is not a principle about languages. It is a principle concerning the organization of grammars dealing both with syntax and semantics. The arguments given above for adhering to the principle, are not based on phenomena in languages, but on properties of grammars satisfying the framework. If one is not pleased with the power of the grammars, one might formulate severe restrictions within the framework. In the light of the examples to be given in chapter 5, it seems that the framework as it is, gives, from a practical viewpoint, already more than enough restrictions.

CHAPTER II

THE ALGEBRAIC FRAMEWORK

ABSTRACT

In this chapter a formal framework is defined for the description of the syntax and semantics of languages. The theory of many-sorted algebra which is needed for this framework is explained, and special attention is paid to the motivation and mathematical justification of the framework. The framework is a synthesis of the approaches of Montague and Adj and it constitutes a formalization of the principle of compositionality of meaning.



1. INTRODUCTION

The aim of this chapter is to present a mathematical description of a framework for the description of syntax and semantics of a language. The framework is a formalization of the principle of compositionality of meaning. The framework is based upon universal algebra: a branch of mathematics which is concerned with the general theory of algebraic structures (the standard work in this field is GRAETZER 1968). Universal algebra deals with the general structures we need, and it provides a language which allows us to speak with precision about such abstract structures. The most important contribution of universal algebra to this book consists of the concepts it provides. I will hardly use any deep mathematical results from universal algebra, but mainly rather elementary notions such as 'subalgebra', 'homomorphism' and 'polynomial' (here generalized to the case of many sorted algebras).

The framework I will present is designed with two predecessors in mind: 'Universal Grammar' (MONTAGUE 1970b), and 'Initial algebra semantics' (ADJ 1977). Montague did not use many sorted algebras, although it is the natural mathematical notion for his purposes. The group Adj was not primarily interested in developing a general framework, but in its practical applications. The present framework is based upon the ideas of Montague, and on the techniques of Adj, and as such it is new. In a few cases, a definition or theorem concerning this framework deviates considerably from what can be found in the literature. The present framework is developed for practical purposes, and I constantly kept PTQ (MONTAGUE 1973) and its successors in mind. As often happens in applying mathematics, the available theory was not applicable in its original form. I had to invent definitions myself, with the literature as a source of analogous notions (this point is also made in Van BENTHEM 1979a,p.17). In the presentation much attention is paid to the motivation of the definitions: if one understands why definitions are the way they are, then it is possible to predict what happens when the conditions in the definitions are violated. The insights developed in this chapter will also be useful in the discussion of several deviations from the framework (see chapter 5). My aim is to give a comprehensible description of an elegant, very abstract mathematical system. In one respect this attempt probably has not been successful: the description of how to obtain new algebras out of old ones. There is no general theory which I could use here, and I had to apply 'ad hoc' methods (see sections 6 and 7).

2. ALGEBRAS AND SUBALGEBRAS

In chapter 1 it was explained that the key notions in our formalization of the compositionality principle are the notions 'many-sorted algebra' and 'homomorphism'. For several reasons these definitions have to be refined. The definition of 'many sorted algebra' is given below; the definition of 'homomorphism' will be given in section 6.

2.1. DEFINITION. A *many-sorted algebra* of *signature* (S,Γ,τ) is a pair $<(A_s)_{s\in S},(F_\gamma)_{\gamma\in\Gamma}>$ such that

a) S is a non-empty set; its elements are called *sorts*.

b) $(A_s)_{s\in S}$ is an indexed family of sets. The set A_s is called the *carrier* of sort s.

c) Γ is a set; its elements are called *operator indices*.

d) τ is a function such that

$\tau\colon \Gamma \to \bigcup_n S^n \times S$ where $n \in \mathbb{N}$ and $n > 0$.

Thus the function τ assigns to each operator index γ a pair $<w,s>$, where s is a sort, and $w = <s_1,\ldots,s_n>$ is an n-tuple of sorts. Such a pair denotes the type of the operator with index γ. Therefore the pair is called an *operator type*, and the function τ is called a *type assigning function*.

e) $(F_\gamma)_{\gamma\in\Gamma}$ is an indexed family of *operators* such that if

$\tau(\gamma) = <<s_1,\ldots,s_n>,s_{n+1}>$ then $F_\gamma\colon A_{s_1}\times\ldots\times A_{s_n} \to A_{s_{n+1}}$

2.1. END

The definition given above is due to J. Zucker (pers.comm.); it is very close to the one given in ADJ 1977. The main difference is that we have no restrictions on the carriers: they may have an overlap, be included in each other or some may be equal. Another, minor, difference is that we have no nullary operators (i.e. it is not allowed in clause d) that n=0). For a motivation and discussion of these differences, see sections 8 and 9. Structures like many sorted algebras are introduced, under different names, in BIRKHOFF & LIPSON 1970 (heterogeneous algebras) and HIGGINS 1963 (algebras with a scheme of operators).

The different components of the above definition are illustrated in the following example

2.2. EXAMPLE. I describe an algebra E consisting of some English words and sentences.

a) The set of sorts S = {Sent, Term, Verb}

b) The carriers are

E_{Term} = {*John, Mary*}
E_{Verb} = {*run*}
E_{Sent} = {*John runs, Mary runs*}

c) The set of operator indices $\Gamma = \{1\}$

d) The type assigning function τ is defined by $\tau(1)$ = <<Term,Verb>,Sent>.

e) The set of operators is $\{F_1\}$. This operator consists of first adding an *s* to its second argument, followed by a concatenation of its first argument with its thus changed second argument. So

$F_1(\alpha,\beta) = \alpha\ \beta s$ (e.g. F_1(*John, run*) = *John runs*)

2.2. END

In this example I have followed the definition in order to illustrate the definition. It is, however, not the most efficient way to represent the required information. There are several conventions which facilitate these matters. The sorts may be used to denote the carriers as well: we will write $a \in s$ instead of $a \in A_s$. We often will write A when $(A_s)_{s\in S}$ or $\cup_{s\in S}(A_s)$ is meant, but we will use A for the algebra itself as well. By a Σ-algebra we understand an algebra with signature Σ. We will avoid to mention S,Γ and τ when they become clear from the context (or are arbitrary). These conventions are employed in the example which will be given below. A final remark about the notation in MONTAGUE 1970b. There an algebra is denoted as $\langle A_s,F_\gamma\rangle_{s\in S,\gamma\in\Gamma}$. I agree with LINK & VARGA (1975) that this is not a correct notation for what is intended: an algebra is not a collection of pairs, but a pair consisting of two collections (the carriers and the operators).

2.3. EXAMPLE. The algebra <A,F> is defined as follows.

Its sorts are

Nat = {0,1,2,3..} (the natural numbers)
Bool = {true,false} (the truth values)

Its operators are

$F_<$: Nat × Nat → Bool where $F(\alpha,\beta) = \begin{cases}\underline{\text{true}} & \text{if } \alpha < \beta\\ \underline{\text{false}} & \text{otherwise}\end{cases}$

$F_>$: Nat × Nat → Bool where $F(\alpha,\beta) = \begin{cases}\underline{\text{true}} & \text{if } \alpha > \beta\\ \underline{\text{false}} & \text{otherwise}\end{cases}$

So $\langle A_s,F_\gamma\rangle$ is a two sorted algebra with two operators of the same type. S,Γ and τ are implicitly defined by the above description.

2.3. END

It is useful to have some methods to define a new algebra out of an old one. An important method is by means of subalgebras. A subalgebra is, roughly, a collection of subsets of the carriers of an algebra which are closed under the operations of the original algebra. The theorems and definitions which follow, are generalizations of those for the one-sorted case in GRAETZER 1968.

2.4. DEFINITION. Let $A = <(A_s)_{s\in S}, (F_\gamma)_{\gamma\in\Gamma}>$ be an algebra. A *subalgebra* of A is an algebra

$B = <(B_s)_{s\in S}, (F'_\gamma)_{\gamma\in\Gamma}>$

such that

1) for each $s \in S$ it holds that $B_s \subset A_s$
2) for each $\gamma \in \Gamma$ it holds that $F'_\gamma = F_\gamma \restriction B$. (i.e. F'_γ is the restriction of F_γ to B).

2.4. END

Note that from the requirement that B is an algebra, it immediately follows that $F'_\gamma(b_1,\ldots,b_n) \in B$. In the sequel we will not distinguish operators of the original algebra and operators of its subalgebras (e.g. we will not use primes to distinguish them). The next example illustrates this.

2.5. EXAMPLES. Let E be the algebra from example 2.2. Hence E is defined by:

$$E = <(E_s)_{s\in\{Term,Verb,Sent\}},\{F_1\}>$$

where $E_{Term} = \{\textit{John},\textit{Mary}\}$, $E_{Verb} = \{\textit{run}\}$,

$E_{Sent} = \{\textit{John runs}, \textit{Mary runs}\}$

and $F_1 : E_{Term} \times E_{Verb} \to E_{Sent}$

is defined by $F_{(\alpha,\beta)} = \alpha\ \beta s$.

Some examples of subalgebras are:

I. E itself is a subalgebra of E.

II. Let $B_{Term} = \{\textit{John}\}$, $B_{Verb} = \{\textit{run}\}$, and $B_{Sent} = \{\textit{John runs}\}$.
Then $B = <(B_s)_{s\in S},\{F_1\}>$ is a subalgebra of E.

III. Let $C_{Term} = \{\textit{Mary}\}$, $C_{Verb} = \{\textit{run}\}$, and $C_{Sent} = \emptyset$.
Then $C = <(C_s)_{s\in S},\{F_1\}>$ is *not* a subalgebra of E.

IV. Let $D_{Term} = \{John\}$, $D_{Verb} = \emptyset$, and $D_{Sent} = \{Mary\ runs\}$.
Then $D = \langle (D_s)_{s\in S}, \{F_1\}\rangle$ is a subalgebra of E, although a rather strange one.

2.5. END

A sorted collection of subsets of an algebra may be contained in several subalgebras. There is a smallest one among them, namely the intersection of these subalgebras. This is proven in the next theorem, this probably corresponds with the mysterious 'theorem due to Perry Smith' mentioned in MONTAGUE 1973 (p.253).

2.6. THEOREM. *Let* $\langle (B_s^{(i)})_{s\in S}, (F_\gamma)_{\gamma\in\Gamma}\rangle_{i\in I}$ *be a collection subalgebras of the algebra* $\langle (A_s)_{s\in S}, (F_\gamma)_{\gamma\in\Gamma}\rangle$. *For each s we define,* $C_s = \cap_{i\in I}(B_s^{(i)})$. *Then* $\langle C_s, F_\gamma\rangle$ *is a subalgebra of* A.

PROOF. We have to prove that $\langle C_s, F_\gamma\rangle$ is closed under the operations F_γ.
Let $\tau(\gamma) = \langle\langle s_1,\ldots,s_k\rangle, s_{k+1}\rangle$ and $c_1 \in C_{s_1},\ldots,c_k \in C_{s_k}$.
Then for all $i \in I$: $c_1 \in B_{s_1}^{(i)},\ldots, c_k \in B_{s_k}^{(i)}$.
So for all i: $F_\gamma(c_1,\ldots,c_k) \in B_{s_{k+1}}^{(i)}$, and consequently
$F_\gamma(c_1,\ldots,c_k) \in \cap_{i\in I} B_{s_{k+1}}^{(i)} = C_{s_{k+1}}$.
2.6. END

We will often be interested in the smallest algebra containing a given collection of subsets. Then the following terminology is used.

2.7. DEFINITIONS. Let $\langle A,\underline{F}\rangle$ be an algebra, and H a sorted collection of subsets of A. The smallest subalgebra of A containing H is called the *subalgebra generated by* H. This algebra is denoted by $\langle [H],\underline{F}\rangle$, and its elements are denoted by [H]. A sorted collection H of subsets of an algebra $\langle A,\underline{F}\rangle$ is called a *generating set* if $\langle [H],\underline{F}\rangle = \langle A,\underline{F}\rangle$. The elements of the sets in H are called *generators*.
2.7. END

Theorem 2.6. characterizes the algebra $\langle [G],\underline{F}\rangle$ as the intersection of all subalgebras containing G. Another characterization will be given in section 4.

An important consequence of theorem 2.6 is that it allows us to use the power of induction. A property P can be proved to hold for all elements of an algebra $\langle A,\underline{F}\rangle$ by proving that:

1) Property P holds for a generating set G of A
2) The set $B = \{a \in A \mid P(a)\}$ is closed under all $F_\gamma \in \underline{F}$.
   (i.e. if $b_1,\ldots,b_n \in B$ and F_γ is defined for them, then $F_\gamma(b_1,\ldots,b_n) \in B$).

From 2) it follows that $\langle B,\underline{F}\rangle$ is a subalgebra of $\langle A,\underline{F}\rangle$. From 1) it follows that $G \subset B$, hence $\langle [G], \underline{F}\rangle$ is a subalgebra of $\langle B,\underline{F}\rangle$. Since $A = \langle [G],\underline{F}\rangle$ it follows that $A = \langle B,\underline{F}\rangle$. So for all $a \in A$ property P holds.

Theorem 2.6 provides us an easier way to present subalgebras than the method used in example 2.5. The theorem shows that it is sufficient to give a set of generators.

2.8. <u>EXAMPLES</u>. The subalgebra mentioned in example 2.5, case II, can be denoted as:

$\langle [\{\textit{John}\}_{\text{Term}}, \{\textit{run}\}_{\text{Verb}}], \{F_1\}\rangle$

where F_1: Term × Verb → Sent is defined by $F_1(\alpha,\beta) = \alpha\ \beta s$.
Note that the sorts of the generators are mentioned in the subscripts.

The subalgebra mentioned in example 2.5, case III, can be denoted as:

$$\langle [\{\textit{John}\}_{\text{Term}}, \{\textit{Mary runs}\}_{\text{Sent}}]\rangle.$$

This algebra is 'generated' in the formal sense; it is however intuitively strange to have a compound expression (*Mary runs*) as generator.
2.8 END

If the 'super' algebra within which we define a subalgebra is clear from the context, we need not to mention this algebra explicitly. This gives a simplification of the presentation of the subalgebra. Such a situation arises when we wish to define some language, i.e. a subset of all strings over some alphabet. In this situation one may conclude from the generators what the elements of the alphabet are, and the 'super' algebra is the algebra with as carrier all finite strings over this alphabet. An example is given below.

2.9. <u>EXAMPLE</u>. We define an algebra N; the carrier of this algebra consists of denotations for numbers. Leading zero's are not accepted, so *700* is an element of N, whereas *007* is not. This algebra is described by:

$$N = \langle [\{\mathit{0},\mathit{1},\ldots,\mathit{9}\}_{Num}],\{F\}\rangle$$

where F: Num × Num → Num is defined by

$$F(\alpha,\beta) = \begin{cases} \beta & \text{if } \alpha = \mathit{0} \\ \alpha\beta & \text{otherwise.} \end{cases}$$

Now it is implicitly assumed that N determines a subalgebra of

$$A = \langle \{\mathit{0},\mathit{1},\ldots,\mathit{9}\}^{*}_{Num},\{F\}\rangle$$

where F is as just defined, and $\{\mathit{0},\mathit{1},\ldots,\mathit{9}\}^{*}$ is the set of all strings formed of symbols in the set $\{\mathit{0},\mathit{1},\ldots,\mathit{9}\}$.

The difference between A and N is that A contains all strings (*007* included), whereas this is not the case for N. Notice that N is highly ambiguous in the sense that its elements can be obtained from the generators in several ways

e.g. $F(\mathit{1},\mathit{7}) = \mathit{17}$ but also $F(\mathit{0},F(\mathit{1},\mathit{7})) = \mathit{17}$.

2.9. END

The above example concerns an algebra with only one sort. In 2.2 and 2.5 we defined algebras with several sorts, and when we consider a subalgebra of such algebras, we can also avoid writing explicitly the 'super' algebra. In that case the most simple algebra we may take as the 'super' algebra, is the one in which all carriers consist of all possible finite strings.
An example as illustration:

2.10 EXAMPLE. We define an algebra M for number denotations, in which leading zero's are accepted, and in which each element can be obtained from the generators in only one way.

$$\underline{M} = \langle [\{\mathit{0},\mathit{1},\ldots,\mathit{9}\}_{dig}],\{F_1,F_2\}\rangle$$

where F_1: dig → num is defined by $F_1(\alpha) = \alpha$
and F_2: num × dig → num is defined by $F_2(\alpha,\beta) = \alpha\beta$.

So F_1 says that all digits are number denotations, and F_2 says that one obtains a new denotation by concatenating the old denotation with a digit (in this order).

$$\text{e.g. } F_2(F_1(\mathit{7}),\mathit{1}) = \mathit{71} \quad \text{and} \quad F_2(F_2(F_1(\mathit{0}),\mathit{0}),\mathit{7}) = \mathit{007}.$$

The implicit 'super' algebra is

$$<\{\{\mathit{0},\mathit{1},\ldots,\mathit{9}\}^*_{dig},\{\mathit{0},\mathit{1},\ldots,\mathit{9}\}^*_{num}\},\{F_1,F_2\}>$$

2.11. <u>EXAMPLE</u>. Another algebra for number denotations is one which differs from the above one in only one respect. Digits are concatenated with numbers for obtaining new numbers (and not in the opposite order as in 2.10).

$\underline{M}' = <[\{\mathit{0},\mathit{1},\ldots,\mathit{9}\}_{dig}],\{F_1,F_3\}>$
where F_1: dig $\rightarrow$ num defined by $F_1(\alpha) = \alpha$
and F_3: num $\times$ dig $\rightarrow$ num defined by $F_3(\alpha,\beta) = \beta\alpha$.

2.11.END

In the examples 2.8/2.11 subalgebras are defined by mentioning a generating set. In all these examples this was a special set: one which was minimal in the sense that none of the generators can be obtained from the other generators. The following terminology can be used to describe this situation.

2.12. <u>DEFINITIONS</u>. A collection B of generators of algebra $<A,\underline{F}>$ is called A-*independent* if for all $b \in B$ holds that $b \notin <[B-\{b\}],\underline{F}>$.
An algebra $<A,\underline{F}>$ is called *finitely generated* if $A = <[B],\underline{F}>$ where B is some finite A-independent generating set of A.
An algebra is called *infinitely generated* if it is not finitely generated.
A collection of generators G is called *the generating set* of the algebra if the algebra is generated by that set and if all generating collections contain G as subcollection.
2.12. END

3. ALGEBRAS FOR SYNTAX

In most examples we have considered, the carriers consisted of strings of symbols. Such algebras can be used to describe languages, and in fact we did so in the previous section. The set we were interested in was the carrier of a certain sort. This should explain the following definition (the epithet 'general' will be dropped in a more restrictive variant).

3.1. DEFINITION. A *general algebraic grammar* G is a pair <A,s>, where A is a many-sorted algebra, s is a sort of A, and all carriers of A consist of strings over some alphabet. The sort s is called the *distinguished sort*, and the carrier of sort s is called the *language generated* by G, denoted L(G).

3.2. EXAMPLE. Let E be the algebra $<[\{\textit{Mary, John}\}_{\mathrm{Term}},\{\textit{run}\}_{\mathrm{Verb}}],\{F_1\}>$, where F_1: Term × Verb → Sent is defined by $F_1(\alpha,\beta) = \alpha\ \beta s$.
Then the general algebraic grammar <E,Sent> generates the language {*Mary runs, John runs*}; and the general algebraic grammar <E,Verb> generates the language {*run*}.
3.2. END

First a warning. In the French literature one finds the notion 'grammaire algebraique'. This notion has nothing to do with the algebraic grammars we will consider here: 'grammaire algebraique' means the same as 'context free grammar'. In the definition above, I have not used the name algebraic grammar because I will use it for a subclass of the general algebraic grammars. Most general algebraic grammars are not interesting because they do not provide the information needed to generate expressions of the language. This is illustrated by the trivial proof of the statement that there is for any language L (even for non-recursively enumerable ones) a general algebraic grammar generating L. Take the grammar which has as algebra the one with no operators, one sort and L as carrier of that sort. This is an uninteresting grammar. It is not sufficient to add the requirement 'finitely generated'; this is illustrated by the following example.

3.3. EXAMPLE. Let L be some nonempty language over some finite vocabulary V. Let w be an arbitrary element of L. Consider now algebra A

$$A = \langle [V_{s_1}], \{F_1, F_2\} \rangle$$

where $F_1: s_1 \times s_1 \rightarrow s_1$ defined by $F_1(\alpha,\beta) = \alpha\beta$
and $F_2: s_1 \rightarrow s_2$ defined by $F_2(\alpha) = \begin{cases} \alpha & \text{if } \alpha \in L \\ w & \text{otherwise.} \end{cases}$

So F_1 generates all strings over V, whereas F_2 selects those strings which belong to L. The definition of an algebra requires that F_2 be a function, so that F_2 delivers some element of sort s_2 even in case its argument is not in L. For this purpose we use the expression w from L. Now $\langle A,s_2 \rangle$ is a finitely generated algebraic grammar with generated language L.
3.3. END

In case L is empty, then we may take a grammar with an empty generating set: $\langle\langle [\emptyset_{s_1}], \{F_1,F_2\}\rangle, s_1\rangle$.

The crux of the above example lies in the operation F_2. There is no algorithm which for every argument yields its image under F_2. So the general algebraic grammar does not provide us with the information which allows us to generate expressions of the language. The absurdity of such a grammar becomes evident if we replace F_2 by the function F_3:

$F_3(\alpha) = \alpha$ if α is an English sentence, and $F_3(\alpha) = w$ otherwise.

The above example shows that for certain generalized algebraic grammars there exists no algorithm which produces the expressions of the language defined by the grammar. The example also illustrates that such grammars are uninteresting for practical purposes. Therefore we will restrict our attention to those algebraic grammars for which there is an algorithm for producing the expressions of the grammar. For this purpose I require that the operators of the grammars be recursive (the notion 'recursive' is the formal counterpart of the intuitive notion 'constructive', see e.g. ROGERS 1967). But this requirement is not sufficient: a grammar might have only recursive operators, whereas the definition of the set of operators is not recursive. Then we would not know of an arbitrary operator whether it is an operator of the grammar, i.e. we do not effectively have the tools to produce the expressions of the language of that grammar, although the tools themselves are recursive. Therefore I also require that there exists some recursive function which decides whether any given operator belongs to the set of operators of the grammar, in other words, that the set of operators be recursive. For similar reasons it is required that the sets of sorts and generators be recursive. In section 5 it will be proven that for such grammars there indeed exists an algorithm generating the expressions of the language defined by the grammar. A more liberal notion is 'enumerable algebraic grammar'. Also for these grammars there exists a generating algorithm, but they have some unattractive properties (e.g. it is undecidable whether a given derivational history is one from a given grammar. Formal definitions concerning recursivity and algebras are given in e.g. RABIN 1960, but the intuitive explication given above is sufficient for our purposes.

3.4. DEFINITION. An *algebraic grammar* is a general algebraic grammar such that

1. its set of operators and its set of sorts are recursive, and it has a recursive generating set
2. all its operators are recursive.

3.5. DEFINITION. An *enumerable algebraic grammar* is a general algebraic grammar such that

1. its set of operators and its set of sorts are recursively enumerable, and it has a recursively enumerable generating set
2. all its operators are recursive.

3.6. DEFINITION. A *finite algebraic grammar* is an algebraic grammar such that

1. its set of operators, and its set of sorts are finite, and it has a finite generating set
2. all its operators are recursive.

3.6. END

I have formally described what kind of language definition device we will use. Next it will be investigated whether our device restricts the class of languages which can be dealt with. The theorem below is of great theoretical impact. It says that, even when using finite algebraic grammars we can deal with the same class of languages as can be generated by the most powerful language definition devices (Turing machines, Chomsky type 0 languages, van Wijngaarden grammars, recursive functions). This means that the requirement of using an algebraic grammar, which was one of the consequences of the compositionality principle, is a restriction only on the organisation of the syntax, but not on the class of languages which can be described by means of an algebraic grammar. The theorem, however, is, from a practical point of view not useful because it does not help us in any way to find a grammar for a given language; this appears from the fact that the proof neglects all insights one might have about the structure of the language: the sorts in the proof have nothing to do with intrinsic properties of the language under consideration.

3.7. THEOREM. *For each recursively enumerable language over some finite alphabeth there exists a finite algebraic grammar that generates the same language.*

PROOF. Let G be a type-0 grammar. So, following the definition in HOPCROFT & ULLMAN (1979, p. 79) we have

$$G = (V_N, V_T, P, S)$$

where V_N, V_T, P, and S are respectively the non-terminal symbols of the grammar, the terminal symbols, the production rules and the start symbol. The set P consists of a finite list of rules $p_1,\dots,p_n$ of the form $\mu \to \nu$ where $\mu \in (V_N \cup V_T)^+$ and $\nu \in (V_N \cup V_T)^*$; so μ is a nonempty string of symbols over $V_N \cup V_T$, and ν is a possibly empty string over this set.

We have to prove that there is a finite algebraic grammar A such that L(A) = L(G). I distinguish two cases. I) $L(G) = \emptyset$ and II) $L(G) \neq \emptyset$. In case I we take an algebra with an empty set of generators, and that gives us a finite algebraic grammar. In case II we know that there is at least one expression in L(G). Let $e \in L(G)$. Then the finite algebraic grammar A for L(G) is defined below.

There are three sorts in A:

In : the sort which contains the only generator of the algebra: the symbol S.
Mid: the sort of intermediate expressions
Out: the sort of resulting expressions; i.e. the sort of the generated language.

The operations of the algebra will simulate derivations of G. The symbol \$ is used to focus our attention on that part of the string on which an operator of algebra A is applied which simulates some rule of G. If α is some string, we understand by α' the result of deleting from α all occurrences of \$.

The algebra A is defined as follows:

$$A = \langle\langle [S_{In}], (F_\gamma)_{\gamma \in \Gamma}\rangle, Out\rangle$$
$$\text{where } \Gamma = \{1,2,3,4\} \cup P.$$

The operators are defined as follows:

F_1: In $\rightarrow$ Mid

$F_1(\alpha) = \$\alpha$

F_2: Mid $\rightarrow$ Mid

$F_2(\alpha_1\$v\alpha_2) = \alpha_1 v\$\alpha_2$ where $\alpha_1,\alpha_2 \in (V_N \cup V_T)^*$ and $v \in V_N \cup V_T$

$F_2(\alpha) = \alpha$ if α is not of the form $\alpha_1\$v\alpha_2$

F_3: Mid $\rightarrow$ Mid

$F_3(\alpha_1 v\$\alpha_2) = \alpha_1\$v\alpha_2$ where $\alpha_1,\alpha_2 \in (V_N \cup V_T)^*$ and $v \in V_N \cup V_T$

$F_3(\alpha) = \alpha$ if α is not of the form $\alpha_1 v\$\alpha_2$

F_4: Mid $\rightarrow$ Out

$F_4(\alpha) = \alpha'$ if $\alpha \in (V_T \cup \{\$\})^*$, so F_4 deletes the occurrences in α of \$

$F_4(\alpha) = e$ if $\alpha \notin (V_T \cup \{\$\})^*$; remember that $e \in L(G)$

F_{p_i} : Mid $\rightarrow$ Mid

$F_{p_i}(\alpha_1\$\mu\alpha_2) = \alpha_1\$\nu\alpha_2$ where p_i is $\mu \rightarrow \nu$

$F_{p_i}(\alpha) = \alpha$ if α is not of the form just mentioned.

Note that F_{p_i} is a function since the \$-mark indicates to which expression p_i is applied.

The proof that L(G) is generated by this grammar follows from the two lemmas below. But first some definitions.

W	is the set of all finite strings over $V_N \cup V_T \cup \{\$\}$ in which at most one \$ symbol occurs
$W_\$$	is the subset of W of strings in which precisely one \$ symbol occurs

$\alpha \underset{A}{\Rightarrow} \beta$ iff $\beta = F_\gamma(\alpha)$ for some $\gamma \in \Gamma$

$\alpha \underset{G}{\Rightarrow} \beta$ iff $\alpha = \delta\mu\varepsilon$, $\beta = \delta\nu\varepsilon$ and $\mu \rightarrow \nu \in P$, where $\delta, \varepsilon \in (V_N \cup V_T)^*$

$\overset{*}{\underset{A}{\Rightarrow}}$ is the transitive and reflexive closure of $\underset{A}{\Rightarrow}$

$\overset{*}{\underset{G}{\Rightarrow}}$ is the transitive and reflexive closure of $\underset{G}{\Rightarrow}$

Recall that we defined α' as the result of deleting all \$ marks from α.

<u>LEMMA</u>. $L(G) \subset L(A)$.

PROOF. First we prove that $\alpha' \underset{G}{\Rightarrow} \beta'$ implies $\alpha \overset{*}{\underset{A}{\Rightarrow}} \beta$ for all $\alpha,\beta \in W_\$$.
Consider the following three cases:

1. $\alpha' = \beta'$ and $\alpha = \beta$. Then $\alpha \overset{*}{\underset{A}{\Rightarrow}} \beta$.
2. $\alpha' = \beta'$ and $\alpha \neq \beta$. Then α contains a \$ in a different position than in β. By repeated application of F_2 or F_3 the \$ sign can be moved to that position.
3. $\alpha' = \delta\mu\varepsilon$ and $\beta' = \delta\nu\varepsilon$ and $\mu \to \nu \in p_i$, where $\delta, \varepsilon \in (V_N \cup V_T)^*$ So $\alpha \overset{*}{\underset{A}{\Rightarrow}} \delta\$\mu\varepsilon$ (using F_2 or F_3); $\delta\$\mu\varepsilon \underset{A}{\Rightarrow} \delta\$\nu\varepsilon$ (using F_{p_i}) and $\delta\$\nu\varepsilon \overset{*}{\underset{A}{\Rightarrow}} \beta$ (using F_3 or F_2. So $\alpha \overset{*}{\underset{A}{\Rightarrow}} \beta$.

Suppose now that $w \in L(G)$, so $S \overset{*}{\underset{G}{\Rightarrow}} w$. Hence $(\$S)' \overset{*}{\underset{G}{\Rightarrow}} (\$w)'$. Repeated application of the argumentation given above shows that $\$S \overset{*}{\underset{A}{\Rightarrow}} \$w$. Since $S \underset{A}{\Rightarrow} \$S$ and $\$w \underset{A}{\Rightarrow} w$, it follows that $w \in L(A)$.

LEMMA. $L(A) \subset L(G)$.

PROOF. We first prove that $\alpha \underset{A}{\Rightarrow} \beta$ implies $\alpha' \overset{*}{\underset{G}{\Rightarrow}} \beta'$ for all $\alpha,\beta \in W\backslash\{e\}$.
Consider the following five cases

1. $\beta = F_1(\alpha)$. Then $\alpha = S$, $\beta = \$S$ so $\alpha' = \beta'$ hence $\alpha' \overset{*}{\underset{G}{\Rightarrow}} \beta'$
2. $\beta = F_2(\alpha)$. Then $\alpha' = \beta'$.
3. $\beta = F_3(\alpha)$. Then $\alpha' = \beta'$.
4. $\beta = F_4(\alpha)$. Since $\beta \neq e$ we have $\alpha' = \beta'$.
5. $\beta = F_p(\alpha)$ for $p = \mu \to \nu$. Then either $\alpha = \beta$, or $\alpha = \delta\$\mu\varepsilon$ and $\beta = \delta\$\nu\varepsilon$. So $\alpha' \underset{G}{\Rightarrow} \beta'$.

Suppose now that $w \in L(A)$, so $S \overset{*}{\underset{A}{\Rightarrow}} w$. By repeated application of the above argumentation we find that $S \overset{*}{\underset{G}{\Rightarrow}} w$. Hence $L(A)\backslash\{e\} \subset L(G)\backslash\{e\}$. Since $e \in L(G)$ it follows that $L(A) \subset L(G)$.
LEMMA END

From the above two lemmas it follows that $L(A) = L(G)$.
3.7. END

Note that the proof of this theorem does not provide an algorithm for making a finite algebraic grammar for a given type-0 grammar G. The decision whether we are in case I ($L(G) = \emptyset$), or in case II ($L(G) \neq \emptyset$), is not an effective decision because there exists no algorithm which decides whether a given type-0 grammar produces an empty language (HOPCROFT & ULLMAN 1979, p. 218 and p. 189). This non-constructive aspect of the proof is unavoidable, as will be proved in the next section.

We aim at a kind of grammar which produces only recursively enumerable languages. The theorem has as a consequence that in all cases a finite grammar is sufficient for the syntax. Nevertheless, we will, in the following chapters, frequently use infinite grammars. Such a decision is motivated mainly by semantic considerations.

4. POLYNOMIALS

In this section a method will be presented for describing new operators: polynomials. I will first present an example that is based upon high school algebra. Consider the polynomial $7y+1$. This polynomial is a compound symbol that defines a certain function. The value of this function for a given argument is obtained by substituting the argument in the polynomial for the variable y and calculating the outcome. So its value for argument 2 is $7.2+1$, being 15, and for argument 1 it is 8. From the basic operations of multiplication and addition we have built in this way a new operation. The fact that the polynomial contains a multiplication operation is not evident from the notation; it might be emphasized by writing the polynomial as $7.y+1$. In less familiar algebras the operation symbols are not written between their arguments, but in front. Using this function-argument notation the polynomial gets the form $+(.(7,y),1)$. Functions with several arguments are obtained from polynomials with several variables. An example is the polynomial $7y_1+5y_2$, or equivalently $+(.(7,y_1),.(5,y_2))$. It represents a function which has for $y_1=1$ and $y_2=2$ the value 17. In order to let the polynomial denote a unique function on pairs of integers, we need a convention which determines what the first argument is and what the second. The convention is that this corresponds with the indices of the variables. For the last example this means that the value for the pair $(0,4)$ is 20 and not 28.

The notions discussed above are defined abstractly for the one sorted algebras in GRAETZER 1968. Below I will generalize them to the case of many-sorted algebras. The definitions are somewhat more complicated than in the one sorted case because it is not evident what the first argument and what the second argument is of a polynomial like $P_1(y_1)$. The definition is based upon a suggestion of Jim Thatcher (pers.comm.).

4.1. <u>DEFINITIONS</u>. Let $\underline{A} = \langle (A_s)_{s\in S}, (F_\gamma)_{\gamma\in\Gamma}\rangle$ be a many sorted algebra. For each $s \in S$ we introduce a set VAR_s consisting of countably many variables:

$$VAR_s = \{x_{1,s}, x_{2,s}, \dots\} \qquad VAR = \bigcup_{s\in S} VAR_s.$$

For each element a of A we introduce a symbol $\underline{a}$

$$CON_s^A = \{\bar{a} \mid a \in A_s\} \qquad CON^A = \bigcup_{s\in S} CON^A.$$

For each operator F_γ of type $\langle w,s\rangle \in S^n \times S$ we introduce a symbol $\bar{F}_\gamma$

$$Op^A_{\langle w,s\rangle} = \{\bar{F}_\gamma \mid F_\gamma \in (F_\gamma)_{\gamma\in\Gamma} \text{ and } \tau(\gamma) = \langle w,s\rangle\}$$

$$Op^A = \bigcup_{\langle w,s\rangle\in S^n\times S} \left(Op^A_{\langle w,s\rangle}\right).$$

Let $w \in S^n$ be $\langle s_1,\dots,s_n\rangle$. Then we define

$$X^w = \{x_{1,s_1}, x_{2,s_2}, \dots, x_{n,s_n}\}.$$

4.2. <u>DEFINITIONS</u>. Let A be a many-sorted algebra. By $POL^A_{\langle w,s\rangle}$ we understand the set of *polynomial symbols* over A of type $\langle w,s\rangle$. These sets are inductively defined as follows:

I. If $x_{j,s} \in X^w$, then $x_{j,s} \in POL^A_{\langle w,s\rangle}$

II. If $\bar{c}_s \in CON^A_s$, then $\bar{c}_s \in POL^A_{\langle w,s\rangle}$

III. If $\bar{F}_\gamma \in Op^A_{\langle\langle s_1,\dots,s_k\rangle s_{k+1}\rangle}$ and $p_1 \in POL^A_{\langle w,s_1\rangle},\dots,p_k \in POL^A_{\langle w,s_k\rangle}$

then $\bar{F}_\gamma(p_1,\dots,p_k) \in POL^A_{\langle w,s_{k+1}\rangle}$.

The set POL^A of polynomial symbols over A is defined by $POL^A = \{POL^A_{\langle w,s\rangle} \mid w \in S^n, s \in S\}$. The symbols $\bar{c}_s$ are called the parameters of the polynomial symbols.

4.2. END

Definition 4.2 differs from the standard definition by clause II. The clause is required here since our definition of an algebra does not allow for nullary operators (they are used in the same way, to denote a specific element of the algebra). In the sequel I will omit the bar when no confusion is likely; so I will write c and F_γ instead of $\bar{c}$ and $\bar{F}_\gamma$. Furthermore the superscript A will often be omitted.

A measure for the complexity of a polynomial symbol is its height. (Other names for the same notion are complexity or depth).

4.3. DEFINITION. The *height* h of a polynomial symbol p is defined by the following clauses

I. $h(x) = 0$ if x is a variable

II. $h(c) = 0$ if c is a constant

III. $h(F_\gamma(p_1,\ldots,p_k)) = 1 + \max(h(p_1),\ldots,h(p_k))$.

4.3. END

A polynomial symbol $p \in POL^A_{<w,s>}$ determines uniquely a (polynomial) operator p_A of type <w,s> in the following way.

4.4. DEFINITION. Suppose $w = <s_1,\ldots,s_k>$ and $a_1 \in A_{s_1} \ldots a_{s_k} \in A_{s_k}$. Then $p_A(a_1,\ldots,a_k)$ is defined by

I. if $p = x_{j,s}$ then $p_A(a_1,\ldots,a_k) = a_j$.

II. If $p = \bar{c}_s$ then $p_A(a_1,\ldots,a_k) = c_{s,A}$

III. if $p = \bar{F}_\gamma(p_1,\ldots,p_m)$

then $p_A(a_1,\ldots,a_k) = \bar{F}_{\gamma,A}(p_{1,A}(a_1,\ldots,a_k)\ldots,p_{m,A}(a_1,\ldots,a_k))$

4.4. END

The interpretation of a polynomial does not depend on arguments of which the corresponding variable does not occur in the polynomial.

4.5. THEOREM. *If* $x_{i,s_i} \in X^w$ *does not occur in* $p_{<w,s>}$ *then for all* a_{s_i} *and* b_{s_i} *from* A_{s_i} *we have* $p(a_{s_1},\ldots,a_{s_i},\ldots,a_{s_n}) = p(a_{s_1},\ldots,b_{s_i},\ldots,a_{s_n})$.

PROOF. By induction on the height of p.

4.5. END

The following theorem says that the polynomially definable operations give rise to a new algebra over the elements of the old algebra. The operators of the new algebra are the (interpretations of) the polynomial symbols.

4.6. THEOREM. *Let* $A = <(A_s)_{s\in S},(F_\gamma)_{\gamma\in\Gamma}>$ *be an algebra and* $(G_\delta)_{\delta\in\Delta}$ *the collection polynomial symbols over* A. *Then* $B = <(A_s)_{s\in S},(G_{\delta,A})_{\delta\in\Delta}>$ *is an algebra.*

PROOF. Each G_δ defines a function, and $(A_s)_{s\in S}$ is closed under such functions since

I. The polynomials of the form $x_{i,s}$ yield one of the arguments as result.
II. The polynomials of the form c_s yield an element of A_s as result.
III. The collection $(A_s)_{s\in S}$ is closed under the operations F_γ.
4.6. END

Note that for each operator F_γ from A there is a corresponding polynomial symbol G in B such that $F_\gamma = G_A$. Let $\tau(\gamma) = \langle\langle w_1,\ldots,w_n\rangle,w_{n+1}\rangle$. Then the polynomial symbol corresponding to F_γ is $\bar{F}_\gamma(x_{1,w_1},x_{2,w_2},\ldots,x_{n,w_n})$.

4.7. EXAMPLE *Formulas from propositional logic*

In this example several algebras are presented, of which the last one is an algebra defining formulas of propositional logic.
Let $V = \{p,q,r\} \cup \{\neg,\vee,\wedge,\rightarrow,(,)\}$.
Consider
$A = \langle[V_t],C\rangle$, where C is the two-place concatenation operator; so $C(p,\rightarrow) = p\rightarrow$. Let α,β,γ be $x_{1,s},x_{2,s},x_{3,s}$ respectively. Then we define

$$A' = \langle[V_t],\{C_2,C_3\}\rangle$$

where C_2 and C_3 are the 2-place and 3-place concatenation operators:

$$C_2 = C(\alpha,\beta)_{\langle\langle t,t\rangle,t\rangle} \text{ and } C_3 = C(C(\alpha,\beta),\gamma)_{\langle\langle t,t,t\rangle,t\rangle}.$$

Out of this algebra we define a new one:

$$B = \langle[\{p,q,r\}_t],\{R_\wedge,R_\vee,R_\rightarrow,R_\neg\}\rangle$$

where the R's are polynomial symbols over A':

$$R_\neg = C_2(\neg,C_3((,\alpha,))) \qquad \text{so } R_\neg(\alpha) = \neg(\alpha)$$

$$R_\wedge = C_3(C_3((,\alpha,)),\wedge,C_3((,\beta,))) \quad \text{so } R_\wedge(\alpha,\beta) = (\alpha)\wedge(\beta)$$

and analogously for $R_\vee$ and $R_\rightarrow$.
The expressions of B are the formulas from propositional logic with proposition letters p,q and r. One observes that B is step by step defined out

of a very simple algebra: the algebra of all strings over the vocabulary with concatenation as operator. Only on the final level does the algebra provide interesting information concerning the structures of logical expressions. On the final level we may define the meanings of the formulas. Usually one will present only the final algebra, the step by step construction out of the basic algebra is omitted, and the concatenation operators C_2 and C_3 are written as concatenations. An algebra like B will in the sequel be defined as follows:

$$B = <[\{p,q,r\}_t],\{\neg(\alpha),(\alpha)\wedge(\beta),(\alpha)\vee(\beta),(\alpha)\rightarrow(\beta)\}>.$$

4.8. <u>EXAMPLE</u>: *Non-polynomially defined operators*

In example 2.9 we have met an operator which is not a polynomial one. I repeat the relevant aspects of that example. The algebra considered there is one of strings of digits:

$$N = <[\{\mathit{0},\mathit{1},\ldots,\mathit{9}\}_{Num}],\{F\}>$$

where $F\colon Num \times Num \rightarrow Num$ is defined by $F(\alpha,\beta) = \begin{cases}\beta & \text{if } \alpha \equiv 0\\ \alpha\beta & \text{otherwise.}\end{cases}$

The operator F is not defined using some polynomial symbol, i.e. it is not a polynomial operator. By using the if-then-else construction, well known from programming languages, we obtain something of the format of a polynomial:

$$F = \mathit{if}\ \alpha = \mathit{0}\ \mathit{then}\ \beta\ \mathit{else}\ \alpha\beta.$$

This is a convenient way to write the definition in one line, and I will use that notation in the sequel. One might be tempted to think that it becomes a polynomial if one rewrites it in the function argument notation:

$$F = \mathit{if\text{-}then\text{-}else}\ (\alpha{=}\mathit{0},\beta,\alpha\beta).$$

This is, however, not the case. The *if-then-else* operator requires as first argument a truthvalue. So an algebra over which *if-then-else* can be an operator, has to contain the sort of truth values, and operations yielding truth values (e.g. the two-place predicate =). Since the algebra N of number denotations does not contain these, F cannot be a polynomial operator over N. Nevertheless, F is a fully legitimately defined operator in N.

Some other examples of non-polynomial operators are

G_1: take the reversed sequence of symbols, so G_1(*792*) = *297*

G_2: take the digits in even position and concatenate them, so G(*2345*) = *35*

G_3: substitute 7 for each occurrence of 3, so G_3(*3723*) = *7727*.

4.8. END

5. TERM ALGEBRAS

In this section the notion 'term algebra' will be introduced. The carriers of a term algebra consist of polynomial symbols which can be considered as representations of the productions of a generated algebra. Term algebras play an important role in the formalization of the compositionality principle. In chapter 1, section 3, it was explained that the meaning of an expression depends on its derivational history. A term algebra represents derivational histories, therefore the meanings of the elements of $A = \langle [(B_s)_{s\in S}],(F_\gamma)_{\gamma\in\Gamma}\rangle$ will be defined on the elements of the corresponding term algebra. Another important aspect of the notion term algebra is that it allows us to describe generated algebras in a way that is more constructive than the description given in section 2 (there they are defined by means of the intersection of a - possibly infinite - number of algebras). The new description will be used to obtain an algorithm generating the elements of an algebra, thus justifying the name 'generated algebra'.

Two arguments are mentioned above for considering generated algebras: semantic interpretation and syntactic production. This means that, in this context, we do not deal with algebras as such, but with algebras with a specified set of generators. Therefore we introduce the notion of a Σ,X-algebra, being a Σ-algebra with as collection of generators the sorted collection X. The term algebra $T_{\Sigma,X}$ consists of polynomial symbols which contain no variables and which have only parameters that correspond with elements in X.

5.1. DEFINITIONS. A Σ,X-*algebra* A is a Σ-algebra such that $A = \langle [X],\underline{F}\rangle$. Let us assume that $X = (X_s)_{s\in S}$ and $\underline{F} = (F_\gamma)_{\gamma\in\Gamma}$. Then we define the *term* *algebra* $T_{\Sigma,X}$ as the algebra:

$$<(T_{\Sigma,X,A,s})_{s\in S},(F^T_\gamma)_{\gamma\in\Gamma}>$$

where

I. $T_{\Sigma,X,A,s} = \{p \in POL^A_{<w,s>} \mid$ p contains no variables and for all constants $\bar{c}$ in p holds $c \in X\}$

and

II. If $\tau(\gamma) = <<s_1,\ldots,s_n>,s_{n+1}>$ and $t_1 \in T_{\Sigma,X,A,s_1},\ldots,t_n \in T_{\Sigma,X,A,s_n}$ then $F^T_\gamma(t_1,\ldots,t_n) = \bar{F}_\gamma(t_1,\ldots,t_n)$.

5.1. END

In the sequel we will often simplify the notation for the term algebra. We will attach to T a subscript which identifies the intended term algebra sufficiently. For instance, if in the context the algebra A is given with a specified collection of generators, we may write T_A.

5.2. EXAMPLE. Consider the algebra from example 2.11:

$$M = <[\{\mathit{0},\mathit{1},\ldots,\mathit{9}\}_{dig}],\{F_1,F_2\}>.$$

Then examples of elements in the term algebra T_M are $\mathit{0},\mathit{1}$, $F_1(\mathit{0})$, $F_2(F_1(\mathit{0}),\mathit{1})$, $F_2(F_2(F_1(\mathit{1}),\mathit{2}),\mathit{3}))$.

5.2. END

The above example shows that each element of T_M represents a way of producing an element of M from the generators by means of successive application of the operators. The following theorem says that all elements of an algebra can be obtained from expressions in the corresponding term algebra, and that only elements of the algebra are obtained in this way (for the definition of t_A, see def.4.4).

5.3. THEOREM. *Let* $A = <[(B_s)_{s\in S}],(F_\gamma)_{\gamma\in\Gamma}>$ *be an algebra. Then* $a \in A_s$ *iff there is some* $t \in T_A$ *such that* $t_A = a$.

PROOF. Let $K_s = \{a \in A_s \mid$ there is some $t \in T_A$ such that $t_A = a\}$.

Since $\{b_{i,s} \mid b_{i,s,A} \in B_s\} \subset T_{A,s}$, we have $B_s \subset K_s$. Hence $(K_s)_{s\in S}$ contains all generators of A.

Next we prove that $K = \langle (K_s)_{s\in S}, (F_\gamma)_{\gamma\in\Gamma}\rangle$ is a subalgebra of A. It suffices to show that $(K_s)_{s\in S}$ is closed under $(F_\gamma)_{\gamma\in\Gamma}$. Let $\tau(\gamma) = \langle\langle s_1,\ldots,s_n\rangle, s_{n+1}\rangle$ and let $a_1 \in K_{s_1},\ldots,a_n \in K_{s_n}$. By definition of K_s we know that there are $t_1 \in T_{A,s_1},\ldots,t_n \in T_{A,s_n}$ such that $t_{1,A} = a_1,\ldots,t_{n,A} = a_n$. Define t_{n+1} as $\underline{F}_\gamma(t_1,\ldots,t_n)$. Then $t_{n+1,A} = F_{\gamma,A}(a_1,\ldots,a_n)$, so $t_{n+1} \in K_{s_{n+1}}$. Hence K is a subalgebra of A.

Since $\langle [(B_s)_{s\in S}], (F_\gamma)_{\gamma\in\Gamma}\rangle$ is the smallest algebra containing B_s, it follows that K = A, and in particular $K_s = A_s$.

5.3. END

This theorem gives the justification for the algorithm used in the next theorem.

5.4. THEOREM. *Let* A *be an enumerable algebraic grammar. Then there is an algorithm that produces for each sort* s *of* A *the elements of* A_s.

PROOF. Since the grammar is enumerable, there is an algorithm that produces the operators of A, and an algorithm that produces the generators of A. Let Alg_{op} and Alg_{gen} be two such algorithms. The algorithm generating the sets A_s uses these two algorithms.

The algorithm that produces for each sort s of A the elements of A_s can be considered as consisting of a sequence of stages, numbered 1,2,... . Stage N is described as follows.

Perform the first N steps of the algorithm Alg_{op}, thus obtaining a sorted collection of operators, called F_N. Perform the first N steps of the algorithm Alg_{gen}, thus obtaining a sorted collection of generators, called $B_N^{(0)}$. Since we performed a finite number of steps of Alg_{op} and Alg_{gen}, there are finitely many elements in F_N and $B_N^{(0)}$. So for an $f \in F_N$ there are finitely many possible arguments in $B_N^{(0)}$. Perform all these applications of operators in F_N to arguments in $B_N^{(0)}$. In this way finitely many elements are produced. By addition of these elements to $B_N^{(0)}$ we obtain $B_N^{(1)}$. Next we apply each $f \in F_N$ to all possible arguments in $B_N^{(1)}$, add the new elements to $B_N^{(1)}$, etc. This process is repeated until we have obtained $B_N^{(N)}$. This completes the description of stage N, next stage N+1 has to be performed. Notice that N is used three times as a bound: for Alg_{op}, for Alg_{gen} and for the height of the produced polynomials.

The algorithm is rather inefficient: in stage N+1 all elements are produced again which were already produced in stage N. A more efficient algorithm might be designed as a variant of the above algorithm. Our aim, however, was to prove the existence of a generating algorithm, and not to design an efficient one. From the description of the algorithm it should be clear that only elements of the algebra are produced.

It remains to be proven that the algorithm produces all elements of A. Theorem 5.3 says that for each $a \in A$ there is a term t in the corresponding term algebra such that $t_A = a$. For each term t there is a stage in the above algorithm in which t_A is produced for the first time. This appears from considering the following two cases.

I. t is a generator:
Since Alg_{gen} produces all generators of A, there is a number N_t such that after N_t steps t_A is produced.

II. $t = F_\gamma(t_1,\ldots,t_n)$.
Assume that $t_{1,A},\ldots,t_{n,A}$ are produced for the first time in stages $N_{t_1},\ldots,N_{t_n}$ respectively, and that F_γ is produced in stage N_{F_γ}. Then t_A is produced in stage $\max(N_{F_\gamma},N_{t_1},\ldots,N_{t_n})+1$.

5.4. END

Theorem 5.4 says that an enumerable grammar produces a recursively enumerable language. In theorem 3.7 it is proven that every recursively enumerable language over a finite alphabet can be produced by a finite algebraic grammar. So every enumerable algebraic grammar (and every algebraic grammar) over a finite alphabet can be 'replaced' by a finite algebraic grammar (this observation is due to Johan van Benthem). As has been said, our choice of a grammar depends also on semantic considerations, and these might lead us to the use of an enumerable grammar instead of a finite one.

The proof of theorem 3.7 contains a non-constructive step. This cannot be avoided, as follows from the next theorem.

5.5. THEOREM. *Let* A *be a finite* Σ,X-*grammar. Then for each sort* s *of* A *it is decidable whether* $A_s = \emptyset$.

PROOF. The algorithm proceeds as follows:
stage 1:
For all sorts s check whether there is a generator of sort s, i.e. whether $X_s = \emptyset$. If there are no generators at all, then all carriers are empty, and the algorithm halts here. If generators are found, then it follows that the corresponding sorts have non-empty carriers.

stage 2N:
For all operators F_γ we check whether they give us about new sorts the information that they are non-empty. Assume $\tau(\gamma) = \langle\langle s_1,\ldots,s_n\rangle,s_{n+1}\rangle$. If it was shown in a previous stage that $A_{s_1},\ldots,A_{s_n}$ are non-empty, then it follows that $A_{s_{n+1}}$ is non-empty as well.
stage 2N+1:
If with the results of the previous stage all carriers are shown to be non-empty, then the algorithm halts here. If in the previous stage no new sort was found with a non-empty carrier, then it follows that all remaining carriers are empty, and the algorithm halts here. If in the previous stage some new sort was found with a non-empty carrier, then go to stage 2N+2 (which is described above).
5.5. END

In theorem 3.7 it is stated that every recursively enumerable language over some finite alphabet can be produced by means of a finite algebraic grammar. The proof was based upon the construction of a finite algebraic grammar simulating a type-0 grammar. That construction was not effective: the construction depends on the question whether the type-0 grammar produces an empty language or not, which question is undecidable (HOPCROFT & ULLMAN 1979, p.281). Every constructive version of theorem 3.7 would reduce emptyness of type-0 languages to emptyness of algebraic grammars. Since the emptyness of algebraic grammars is decidable (th.5.5), such a reduction is impossible.

In chapter 1 an interpretation of Frege's principle was mentioned which I described as the 'most intuitive' interpretation. It says that the parts of a compound expression have to be visible parts of the compound expression and that a syntactic rule concatenates these parts. The following theorem concerns such grammars. It is shown that we get such grammars as a special case of our framework.

5.6. <u>THEOREM</u>. *Let* A *be a finite algebraic grammar with generating set* B. *Suppose that all operations* A *are of the form*

$$F_\gamma(a_1,\ldots,a_n) = w_{0,\gamma}a_1w_{1,\gamma}a_2w_{2,\gamma}\ldots a_nw_{n,\gamma}$$

where $w_{i,\gamma}$ is some possibly empty string of symbols.

Then L(A) *is a context free language.*

PROOF. We define a context free grammar G as follows:

The set V_N of non-terminal symbols of G consists of symbols corresponding with sorts of A:

$$V_N = \{\bar{s} \mid s \text{ is a sort of } A\}.$$

The start symbol of the grammar G is the symbol corresponding with the distinguished sort s (i.e. the sort such that $L(A) = A_s$).

The set V_T of terminal symbols of G consists of symbols corresponding with the generators of A:

$$V_T = B.$$

The collection of rules of G consists of two subcollections

$$R = \{\bar{s} \to b \mid b \in B_s\} \cup \{\bar{s}_{n+1} \to w_{0,\gamma}\bar{s}_1 w_{1,\gamma}\bar{s}_2 w_{2,\gamma} \ldots \bar{s}_n w_{n,\gamma} \mid \tau(\gamma) = \langle\langle s_1, s_2, \ldots, s_n\rangle, s_{n+1}\rangle\}.$$

It should be clear that $L(G) = L(A)$. This means that $L(A)$ is context free.
5.6. END

The above theorem could easily be generalized to the case that the arguments of an operation are not concatenated in the given order, but are permuted first. Theorem 5.6 shows that the most intuitive interpretation of Frege's principle (all parts have to be visible parts) is a special case of our framework. It shows moreover that with this interpretation one either has to accept an infinite number of operators, or to conclude that the principle can only be applied to context free languages. A restriction to context free languages is not attractive because that would exclude a large class of interesting languages (e.g. ALGOL 68 and predicate logic), furthermore it has been claimed that natural languages are not context free (for a discussion see PULLUM & GAZDAR 1982). An attempt to use only context free rules for the treatment of natural language, but an infinite number of them, is made by GAZDAR (1982).

In our approach the generation of a context-free language is only a special case of the framework. The group Adj, which works in a similar framework, seems to have another opinion about context-freedom. They give no explicit definition of the notion 'algebraic grammar' nor of its 'generated language', but the definition they implicitly use, seems similar to ours. They suggest, however, that by using algebraic grammars, one can obtain only context-free languages. Evidence for this is that they construct an algebraic grammar for a context-free language and next state that that is 'the most important and general example' (ADJ 1977, p.75). Another statement suggesting this arises when they discuss SCOTT & STRACHEY 1971. Those authors say (p.29):

> *Our language .. is no longer context free. But if we may say so, who cares? .. The last thing we want to be dogmatic about is language.*

As a reaction to this, they say (ADJ 1977, p.76)):

> *'But their semantics does depend on the context free character of the source language, because the meaning of a phrase is a function of the meanings of its constituent phrases'.*

So again they take for granted that an algebraic grammar generates a context-free language. The difference of opinion in these matters might be explained by the fact that they consider only a very special relation between the syntactic algebra and the corresponding term algebra (but see also the discussion in section 9).

6. HOMOMORPHISMS

A homomorphism from algebra A to B is a mapping from the carriers of A to the carriers of B such that the structure of A and B is respected. This is only possible if A and B have about the same structure, although it is not needed that A and B are identical or isomorph. For instance, it is not needed that the two algebras have the same sorts, but there has to be a one-one correspondence of the sorts. It is not necessary that the operators perform the same action, but there has to be a one-one correspondence between the operators such that if an operator in A is defined for certain sorts, then the corresponding operator in B is defined for the corresponding sorts in B. These considerations are expressed formally in the following definitions (they are due to J. Zucker, pers. comm.).

6.1. DEFINITION. Let A be an algebra with signature $\Sigma_A = (S_A,\Gamma_A,\tau_A)$, and B an algebra with signature $\Sigma_B = (S_B,\Gamma_B,\tau_B)$. Let $\sigma\colon S_A \to S_B$ and $\rho\colon \Gamma_A \to \Gamma_B$ be bijections (i.e. mappings which are one-one and onto). Then two algebras A and B are called (σ,ρ)-*similar* if the following holds:

$$\tau_A(\gamma) = \langle\langle s_1,\dots,s_n\rangle,s_{n+1}\rangle \quad \text{if and only if}$$

$$\tau_B(\rho(\gamma)) = \langle\langle\sigma(s_1),\dots,\sigma(s_n)\rangle,\sigma(s_{n+1})\rangle.$$

If σ and ρ are fixed in a certain context, we will omit them and say that the algebras A and B are *similar*.

6.2. DEFINITIONS. Let A and B be (σ,ρ)-similar algebras. By a (σ,ρ)-*homomorphism* h from A to B we understand a mapping h: $\cup_{s\in S_A}(A_s) \to \cup_{s\in S_B}(B_s)$ from the carriers of A to the carriers of B such that

1) $h(A_s) \subset B_{\sigma(s)}$.
2) If $\tau_A(\gamma) = \langle\langle s_1,\dots,s_n\rangle,s_{n+1}\rangle$ and $a_1 \in A_{s_1},\dots,a_n \in A_{s_n}$ then $h(F_\gamma(a_1,\dots,a_n)) = F_{\rho(\gamma)}(h(a_1),\dots,h(a_n))$.

The collection of (σ,ρ)-homomorphisms from A to B, where A and B are (σ,ρ)-similar algebras, is denoted Hom(A,B,σ,ρ). When σ and ρ are clear from the context, or are arbitrary (but fixed), then we will simply speak of a homomorphism h; the collection is then denoted by Hom(A,B).

In case h is surjective, it is called a *homomorphism onto*, or an *epimorphism*. The collection of epimorphisms is denoted Epi(a,B,σ,ρ), or simplified Epi(A,B). In case h is bijective (one-one and onto), it is called an *isomorphism* (note that in category theory this term is used with a different meaning).

6.2. END

The definition of 'homomorphism' given in 6.2 differs from the one given by Adj (see e.g. ADJ 1977). One difference is that our definition can be used in more circumstances: we do not require, for instance, that the collections of operator indices and sorts are identical. I prefer, in this respect, our definition for practical reasons. Sometimes algebras have 'natural' sorts, e.g. an algebra generating a language may have a carrier of the sort sentence, whereas a semantical algebra may have a sort of truth-values, or of propositions. Then one might wish to define a homomorphism between these two algebras, although the sorts are not identical. Our definition allows to do so directly, whereas according Adj's definition renaming of the sorts has to be done first. This difference in the definitions is, in theoretical respect, not important, and does not give rise to interesting theoretical consequences. In the following theoretical investigations

I will assume, for the ease of discussion, that similar algebras do have the same sorts and operator indices; then σ and ρ are assumed to be the identity mapping. A more fundamental difference of the definitions is that Adj defines a homomorphism as a sorted collection of mappings $(h_s)_{s\in S}$, where $h_s\colon A_s \rightarrow B_s$, and where these operations respect, in a certain sense, the structure of the algebras involved. Since, according to our definition of a many sorted algebra, the carriers need not be disjoint, it would under Adj's definition of homomorphism be possible for an element occurring in two carriers to have two different images under h. In section 9 it will be explained why Adj's definition is not suitable for us in this respect.

A homomorphism respects structure. Therefore it is not surprising that the homomorphic image of an algebra is a (similar) algebra. This is expressed in the following theorem.

6.3. <u>THEOREM</u>. *Let* $A = \langle\langle A_s)_{s\in S},(F_\gamma)_{\gamma\in\Gamma}\rangle$ *and* $B = \langle(B_s)_{s\in S},(G_\gamma)_{\gamma\in\Gamma}\rangle$ *be similar algebras, and* $h \in \mathrm{Hom}(A,B)$. *Then* $\langle(h(A_s))_{s\in S},(G_\gamma)_{\gamma\in\Gamma}\rangle$ *is a subalgebra of* $\langle(B_s)_{s\in S}(G_\gamma)_{\gamma\in\Gamma}\rangle$.

<u>PROOF</u>. We prove the theorem by proving that the sets $h(A_s)_{s\in S}$ are closed under G_γ. Let $\tau(\gamma) = \langle\langle s_1,s_2,\ldots,s_n\rangle,s_{n+1}\rangle$ and let $b_i \in h(A_{s_i})$. This means that there are $a_i \in A_{s_i}$ such that $b_i = h(a_i)$. Consequently

$$G_\gamma(b_1,\ldots,b_n) = G_\gamma(h(a_1),\ldots,h(a_n)) = h(F_\gamma(a_1,\ldots,a_n)).$$

It is clear from the last expression that it denotes an element of $h(A_{s_{n+1}})$, so of $B_{s_{n+1}}$.
6.3. END

In chapter 1 we discussed the way in which the set E of expressions of the language should be related to the set D of semantic objects. We concluded that, in order to obey the compositionality principle, the syntax has to be a many sorted algebra and that the meaning of an expression has to be obtained in the following way. For each syntactic operator F_γ, there is an operator G_γ on D, where G_γ is defined for the images of the arguments of F_γ. For the mapping M which yields the corresponding meaning it is required that:

$$M(F_\gamma(e_1,\ldots,e_k) = G_\gamma(M(e_1),\ldots,M(e_k)).$$

We concluded that these requirements have the consequence that D gets the same structure as the syntactic algebra. More formally this is stated in the following theorem.

6.4. <u>THEOREM</u>. *Let* $E = \langle (E_s)_{s\in S}, (F_\gamma)_{\gamma\in\Gamma}\rangle$ *be an algebra,* D *a set and* M *a mapping from* E *to* D. *Let* $(G_\gamma)_{\gamma\in\Gamma}$ *be operators defined on the subsets* $M(E_s)$ *of* D.
Suppose

$$M(F_\gamma(e_1,\ldots,e_k)) = G_\gamma(M(e_1),\ldots,M(e_k))$$

for all $\gamma \in \Gamma$ *and for all arguments* $e_1,\ldots,e_k$ *for which* F_γ *is defined.*
Then $D' = \langle (M(E_s))_{s\in S}, (G_\gamma)_{\gamma\in\Gamma}\rangle$ *is an algebra similar to* E.

<u>PROOF</u>.
I. $(D'_s)_{s\in S}$ is a collection of sets closed under the operations G_γ since $G_\gamma(m_1,\ldots,m_k) = G_\gamma(M(e_1),\ldots,M(e_k)) = M(F_\gamma(e_1,\ldots,e_k)) \in D'_{s_{k+1}}$.
II. D is similar to E since the sorts are the same, the operator indices are the same and
if $F_\gamma : E_{s_1} \times E_{s_2} \times \ldots E_{s_n} \to E_{s_{n+1}}$
then $G : D'_{s_1} \times D'_{s_2} \times \ldots D'_{s_n} \to D'_{s_{n+1}}$.
III. M is a mapping onto $\cup_s D'_s$ satisfying the conditions for homomorphisms.
6.4. END

Having introduced the notion 'homomorphism', we may formalize the compositionality principle as follows: the syntax is a many sorted algebra A, the semantic domain is a similar algebra M, and the meaning assignment is a homomorphism from the term algebra T_A to M.

A first consequence of this formalization is the following theorem concerning the replacement of expressions with the same meaning.

6.5. <u>THEOREM</u>. *Let* $e, e' \in A_s$, *with* $M(e) = M(e')$. *Suppose* $F_\gamma(\ldots, e, \ldots)$ *to be defined. Then* $M(F_\gamma(..,e,..) = M(F_\gamma(..,e',..))$.

<u>PROOF</u>. $M(F_\gamma(..,e,..) = G_\gamma(..,M(e),..) = G_\gamma(..,M(e'),..) = M(F_\gamma(..,e',..))$.
The equalities hold since M is a homomorphism and F_γ is defined for all elements of A_s.
6.5. END

The theorem states that in case two expressions of the same category have the same meaning, they can be interchanged in all contexts without changing the resulting meaning. The reverse is not true, interchangeable in all contexts without changing the meaning does not imply that the meanings are identical, since the language might be too poor to provide for contexts where the difference becomes visible.

The above theorem is related to the well known principle of Leibniz concerning substitutions (GERHARDT, 1890, p.228).

Eadem sunt, quorum unum potest substitui alteri, salva veritate.
This principle is sloppy formulated: it confuses the thing itself with the name refering to it (CHURCH 1956, p.300, QUINE 1960, p.116). It should be read as saying that two expressions refer to the same object if and only if in all contexts the expressions can be interchanged without changing the truthvalue. Let us generalize the principle to all expressions, instead of only referring ones, thus reading 'Eadem sunt' as 'have the same meaning'. Then the above theorem gives us a formalisation of one direction of Leibniz' principle. The other direction can then be considered as a restriction on the selection of a semantical domain. The semantical domain may only give rise to differences in meaning that are expressible in the language.

An important, although very elementary, property concerning homomorphisms is that the compositions of two homomorphisms h and g is a homomorphism again. As defined in chapter 1, the composition which consists in first applying h and next g is denoted $h \circ g$. This has as a consequence that $(h \circ g)(x) = g(h(x))$, note that the order is reversed here. For this reason one sometimes defines $h \circ g$ as first applying g and next h. Adj follows the standard definition, but in order to avoid the change of the order, they have, in some of their papers the convention to write the argument in front of the operator (so $(x)(h \circ g) = ((x)h)g$!). I will use the standard definition ($h \circ g$ means first applying h). The announced theorem concerning composition of homomorphisms is as follows.

6.6. <u>THEOREM</u>. *Let* A,B *and* C *be similar algebras and let* $h \in \mathrm{Hom}(A,B)$ *and* $g \in \mathrm{Hom}(B,C)$. *Then* $h \circ g \in \mathrm{Hom}(A,C)$.

<u>PROOF</u>. Let F_γ, G_γ and H_γ denote operators in A,B and C respectively. Then

$$h \circ g(F_\gamma(a_1,\ldots,a_k)) = g(h(G_\gamma(a_1,\ldots,a_k))) = g(G_\gamma(h(a_1),\ldots,h(a_k))) =$$
$$H_\gamma(g(h(a_1)),\ldots,g(h(a_k))) = H_\gamma(h \circ g(a_1),\ldots,h \circ g(a_k)).$$

6.6. END

In general, it is not necessary to define a homomorphism by stating its values for all possible arguments, since (as I will now show) in the same way as a subalgebra is completely determined by its generators, a homomorphism is completely determined by its values on these generators.

6.7. THEOREM. *Let* $h,g \in \mathrm{Hom}(<[A],(F_\gamma)_{\gamma\in\Gamma}>, <B,(G_\gamma)_{\gamma\in\Gamma}>)$. *Suppose that* $h(a) = g(a)$ *holds for all generators* $a \in A$. *Then* $h(e) = g(e)$ *holds for all elements* e *of* $<[A],(F_\gamma)_{\gamma\in\Gamma}>$.

PROOF. Let $K = \{a \in [A] \mid h(a) = g(a)\}$. Now K is closed under the operations $(F_\gamma)_{\gamma\in\Gamma}$:

Let $k_1,\ldots,k_n \in K$. Then:
$h(F_\gamma(k_1,\ldots,k_n)) = F_\gamma(h(k_1),\ldots,h(k_n)) =$
$F_\gamma(g(k_1),\ldots,h(k_n)) = g(F_\gamma(k_1,\ldots,k_n))$.

So K is a subalgebra with $A \subset K$. Since $[A]$ is the smallest subalgebra with this property, it follows that $K = [A]$.

6.7. END

Suppose that we have defined a mapping from the generators of algebra A to those of algebra B. Then the above theorem says that there is at most one extension of this mapping to a homomorphism. But not in all cases such an extension exists. Suppose that in A two different operators yield for different arguments the same result (i.e. $F_i(a_1,\ldots,a_n) = F_j(a'_1,\ldots,a'_m)$), whereas this is not the case in B for the corresponding operators. Then there is no homomorphism from A to B. But for the algebras we will work with, (viz. termalgebras) this situation cannot arise. In a termalgebra an operator (e.g.S_1) leaves the corresponding symbol as a trace in the resulting expression (e.g. the symbol S_1 in $S_1(E_1,E_2)$). Hence in a termalgebra different operators always yield different results. Therefore we may define a meaning assigning homomorphism by providing 1. meanings for the generators of the syntactic algebra and 2. semantic operators corresponding to the syntactic operators.

6.8. EXAMPLE. Let M be as in example 2.10, so

$M = <[\{\mathit{0,1,2,\ldots,9}\}_{\mathrm{dig}}], \{F_1,F_2\}>$

where F_1: dig $\to$ num $\quad F_1(\alpha) = \alpha$

and F_2: num $\times$ dig $\to$ num $\quad F_2(\alpha,\beta) = \alpha\beta$.

This algebra produces strings of symbols. The meaning of such a string has to be some natural number. Let us denote natural numbers by symbols such as 7, 70 etc. The reader should be aware of the fact that there is (in this example) a great difference between strings such as *1* and *7* for which e.g. concatenation is defined, but not addition or multiplication, and numbers such as 1 and 7 for which addition and multiplication are defined, but not concatenation. Another difference is that *7*,*07*, and *007* are distinct strings all corresponding to the same number 7. The meaning algebra N corresponding to M, consists of numbers and is defined as follows.

$$N = \langle [\{0,1,\dots,9\}_{dig}], \{G_1, G_2\} \rangle$$
$$\text{where } G_1\colon dig \to num \qquad \text{defined by } G_1(\alpha) = \alpha$$
$$\text{and } G_2\colon num \times dig \to num \quad \text{defined by } G_2(\alpha,\beta) = 10 \times \alpha + \beta.$$

The meaning homomorphism h is defined by $h(\mathit{0}) = 0 \dots h(\mathit{9}) = 9$.
So

$$h(F_1(\mathit{1})) = G_1(h(\mathit{1})) = G_1(1) = 1$$

and

$$h(F_2(F_1(\mathit{0}),\mathit{7})) = G_2(G_1(0),7) = 10 \times 0 + 7 = 7.$$

6.9. <u>EXAMPLE</u>. In example 2.11 we considered an algebra which was the same as the above one, with the difference that the digits are written in front of the numbers:

$$F_3\colon num \times dig \to num \qquad \text{defined by } F_3(\alpha,\beta) = \beta\alpha.$$

In this situation it is impossible to find a semantic operation G_3 corresponding with F_3. For suppose there were such an operation G_3. Then, since e.g. $h(\mathit{7}) = h(\mathit{007}) = 7$, we would have that on the one hand $G_3(7,h(\mathit{2})) =$ $= G_3(h(\mathit{7}),h(\mathit{2})) = h(F_3(\mathit{7},\mathit{2})) = h(\mathit{27}) = 27$, but on the other hand $G_3(7,h(\mathit{2})) =$ $= G_3(h(\mathit{007}),h(\mathit{2})) = h(F_3(\mathit{007},\mathit{2})) = h(\mathit{2007}) = 2007$, which is a contradiction. So whereas in example 6.8 it was rather easy to find a semantic operation, it here is impossible since on the level of semantics there is no difference between the meanings of *7* and *007*.
6.9. END

The last example is a formal illustration of a statement of Montague's concerning the syntax of natural languages (MONTAGUE 1970b, p.223, fn.2):

It is to be expected, then, that the aim of syntax can be realized in many different ways, only some of which would provide a suitable basis for semantics.

Next I will prove a theorem that is important from a theoretical point of view. The theorem implies that the framework allows for assigning any meaning to any language. This means that working in accordance with the framework gives rise to no restriction: neither on the produced languages, nor on the assigned meanings. Notice that there is no conflict between the theorem and the example above. The example shows that not every syntax can be used, whereas the theorem states that there is at least one syntax. The theorem is, however, not useful from a practical point of view, since it is based upon the syntax developed in theorem 3.7: the construction does not reflect the structure of the language. The proof of the theorem just says that if you know what the intended meanings of the expressions of the language are, then this knowledge defines some algebraic operation.

6.10. THEOREM. *Let* L *be a recursively enumerable language over a finite alphabet, and* M *a set of meanings for elements of* L. *Let* $f\colon L \to M$ *be a function. Then there is a finite algebraic grammar* A *and an algebra* B *such that*
1) $L(A) = L$.
2) A *and* B *are similar.*
3) *There is an* $h \in \mathrm{Epi}(A,B)$ *such that* $h(w) = f(w)$ *for all* $w \in L$.

PROOF. For the case $L = \emptyset$ the theorem is trivial. For the case $L \neq \emptyset$ consider the algebraic grammar defined in theorem 3.7.

Let A be the algebraic grammar obtained in this way for L.

Recall that the last operation of this algebra gives for some strings as output the same string, but with \$ deleted. For other strings it gives a special string as output. The semantic algebra will differ from the syntactic one only in this last operation: the function f will be incorporated.

More formally, let A be the syntactic algebra from theorem 3.7. So $A = \langle\langle [S_{in}], (F_\gamma)_{\gamma\in P\cup\{1,2,3,4,5\}}\rangle, in\rangle$, where $S_A = \{in, mid, out\}$. This algebra is transformed into a semantic algebra B as follows:

$$B = \langle (B_s)_{s\in S}, (G_\gamma)_{\gamma\in P\cup\{1,2,3,4,5\}}\rangle$$

where $B_{in} = A_{in}$, $B_{mid} = A_{mid}$ and $B_{out} = M$ and

$$G_\gamma = F_\gamma \quad \text{if} \quad \gamma \neq 5,$$

and

$$G_5 = F_5 \circ f.$$

The homomorphism h is defined by

$$h(w) = \begin{cases} w & \text{for } w \in B_{in} \cup B_{mid} \\ f(w) & \text{for } w \in B_{out}. \end{cases}$$

6.10. END

7. A SAFE DERIVER

In chapter 1, section 4, I have sketched the framework in which we will work, and I will repeat here some relevant aspects. The syntax of the language of which we wish to define the semantics is an algebra A, and the function which assigns meanings is an homomorphism defined on T_A. In order to define this homomorphism we use a logical algebra L which is interpreted by homomorphism h in model M. From the algebra L a new algebra L' is defined, using deriver D, where L' is similar with A. The interpretation h for L should determine uniquely an interpretation h' for L'. This situation is represented in figure 1. We will return to this framework in section 8.

$$\begin{tikzcd}
 & T_A \arrow[d] \\
L \arrow[r, dashed, "D"] \arrow[d, "h"] & L' \arrow[d, "h'"] \\
M \arrow[r, dashed, "D"] & M'
\end{tikzcd}$$

Figure 1. The framework

In this and in the next section I will investigate some methods for building new algebras out of old ones, in such a way that an interpretation homomorphism defined on the old algebra determines a unique interpretation for the new algebra. Such a method will be called safe. We will meet several examples of methods to obtain new algebras from old ones; as neutral name for such methods I will use deriver.

7.1. <u>DEFINITION</u>. A deriver is called *safe* if for algebras A and B and all $h \in Epi(A,B)$ there is a unique algebra B' such that for the restriction h' of h to D(A) it holds that $h' \in Epi(D(A),B')$.
7.1. END

The requirement that h' is an epimorphism is important. If we would not require this, B' would in most cases not be unique. An extreme example arises when D(A) is an empty algebra. Then there are infinitely many algebras B' such that $h' \in Hom(D(A),B')$, but only one such that $h' \in Epi(D(A),B')$.

In this section I will consider the aspect of the introduction of new operators. MONTAGUE (1970b) claims that polynomially defined operators are safe. A proof that for many-sorted algebras polynomial extensions are safe, will be given below.

7.2. <u>DEFINITION</u>. Let A = <A,F> be an algebra and G a collection of operator symbols such that for all $g \in G$ there are $s_1,\ldots,s_n,s_{n+1} \in S_A$ such that $g_A : A_{s_1} \times \ldots \times A_{s_n} \to A_{s_{n+1}}$. Then the algebra $<A,F \cup G>$ is denoted $Addop_G$ (<A,F>).

7.3. <u>THEOREM</u>. *If* P *is a collection of polynomial symbols, then* $Addop_P$ *is safe.*

<u>PROOF</u>. Let $A = <(A_s)_{s\in S},(F_\gamma)_{\gamma\in\Gamma}>$ and $B = <(B_s)_{s\in S},(G_\gamma)_{\gamma\in\Gamma}>$ be similar algebras. Suppose that $h \in Epi(A,B)$ and $P \times POL^A$. We define $h: POL^A \to POL^B$ as follows:
I. Each operator symbol $\bar{F}_\gamma$ is replaced by a symbol $\bar{G}_\gamma$.
II. Each constant $\bar{a}$ is replaced by a constant $\bar{b}$, where $b = h(a)$.

Define $A' = Addop_P(A)$, $B' = Addop_{h(P)}(B)$.

We now prove that h is an epimorphism from A' onto B'. That h is surjective follows from the fact that $h \in Epi(A,B)$. Remains to show that $h \in Hom(A',B')$, so that for all new operators, i.e. for all $p \in P$, holds:

$$h(p_{A'}(a_1,\ldots,a_n)) = p_{B'}(h(a_1),\ldots,h(a_n)).$$

This is proved by induction on the complexity of p.

I. $p = x_{j,s}$

$$h(x_{j,s,A'}(a_1,\dots,a_n)) = h(a_j) = x_{j,s,B'}(h(a_1),\dots,h(a_n)).$$

II. $p = \bar{a}$

$$h(\bar{a}_A(a_1,\dots,a_n)) = h(\bar{a}_A) = \bar{b}_B = \bar{b}_B(h(a_1),\dots,h(a_n)).$$

III. $p = \bar{\bar{F}}_\gamma(p_1,\dots,p_m)$

$$\begin{aligned}
&h(p_{A'}(a_1,\dots,a_n)) = h(F_\gamma(p_{1,A'},\dots,p_{m,A'})(a_1,\dots,a_n)) = \\
&= h(F_\gamma(p_{1,A'}(a_1,\dots,a_n)),\dots,p_{m,A'}(a_1,\dots,a_n)) = \\
&= G_\gamma(h(p_{1,A'}(a_1,\dots,a_n)),\dots,h(p_{m,A'}(a_1,\dots,a_n))) = \\
&= G_\gamma(p_{1,B'},\dots,p_{m,B'})(h(a_1),\dots,h(a_n)) = p_{B'}(h(a_1),\dots,h(a_n)).
\end{aligned}$$

Next we prove that B' is unique in the following sense: if $h \in \mathrm{Epi}(A',B')$ and $h \in \mathrm{Epi}(A',D)$, then $D = B'$. This follows from:

1. the carriers of B' and D are equal:

$$B'_s = \{h(a) \mid a \in A'_s\} = D_s$$

2. the operators of B' and D are identical:
   Let $b_1 \in B_{s_1},\dots,b_n \in B_{s_n}$. Then there are $a_1 \in A_{s_1},\dots,a_n \in A_{s_n}$ such that $h(a_i) = b_i$. Hence $p_{B'}(b_1,\dots,b_n) = p_{B'}(h(a_1),\dots,h(a_n)) =$
   $= h(p_{A'}(a_1,\dots,a_n)) = p_D(h(a_1),\dots,h(a_n)) = p_D(b_1,\dots,b_n)$.

One observes that uniqueness is a direct consequence of existence.
7.3. END

Now the question arises whether the restriction to polynomially defined operations is necessary. We cannot generalize theorem 7.3 to operators which are defined in an arbitrary way, as is shown by the next example.

7.4. EXAMPLE. Consider the following algebra of strings of digits:

$$N = \langle \{0,1,\dots,9\}_{dig}, \{0,\dots,9\}^*_{num}, C\rangle$$

where $C : N_{dig} \times N_{num} \to N_{num}$ is defined by $C(\alpha,\beta) = \alpha\beta$.

With these strings we associate a somewhat unusual meaning: their length (it is not that unusual if one remembers the system 1-one, 11-two, 111-three). So the semantic algebra M corresponding to N consists of natural numbers. Notice that in N we had digits (denoted *0*,*1*, etc.) with concatenation, but in M we have natural numbers (denoted 0,1 etc.), with the operation of addition. The interpretation homomorphism h from N to M is defined as follows

$$h(\mathit{0}) = h(\mathit{1}) = .. \; h(\mathit{9}) = 1.$$

The operation corresponding with C is of course addition of the lengths of the strings. So the semantic algebra M is defined as

$$M = \langle (\{1\}_{dig}, \mathbb{N}_{num}), + \rangle.$$

Now we extend N with a new operator D, defined as follows:

$$D\colon N_{dig} \times N_{num} \to N_{num}$$

$$\text{where } D(\alpha,\beta) = \begin{cases} \beta & \text{if } \alpha \text{ is the symbol } 0 \\ \alpha\beta & \text{otherwise.} \end{cases}$$

This operator is not polynomially defined, and it cannot be defined polynomially because there are no truth values in the algebra N (see example 4.8). Let N' be the algebra obtained from N by adding the operator D. Is there a unique algebra M' such that $h \in \mathrm{Epi}(N',M')$?

Suppose that there is such an algebra, called M', with as operator d, corresponding with D. What is then the value of d(1,1)?
On the one hand: $d(1,1) = d(h(\mathit{3}),h(\mathit{7})) = h(D(\mathit{3},\mathit{7})) = h(\mathit{37}) = 2.$
On the other hand: $d(1,1) = d(h(\mathit{0}),h(\mathit{7})) = h(D(\mathit{0},\mathit{7})) = h(\mathit{7}) = 1.$
This is a contradiction. So there is no such algebra M'. The source of this problem is that we make a distinction at the syntactic level which has no influence on the semantic level: the difference between *0* and the other digits.
7.4. END

This example has shown the dangers of using a non-polynomially defined operator. If one introduces an operator which is defined in some arbitrary way, then there is the danger of disturbing the interpretation homomorphism. In practice the situation often arises that the meaning of some language is defined by translation into a logic. The addition to the logic of an operator which is not polynomially defined, could invoke the danger that there is no longer an associated semantics: a translation is defined, but there is no guarantee of an interpretation for the derived logical algebra (i.e. there is, in figure 1, no h'). In chapter 6 (part 2) we will meet several examples of proposals from the literature which are incorrect since there is not such an interpretation.

The following example shows that in some cases operators which are not polynomially definable nevertheless may respect homomorphic interpretations. So in theorem 7.3 the condition 'polynomially defined' is not a necessary condition.

7.5. <u>EXAMPLE</u> (W. Peremans, pers. comm.).

Consider the algebra of natural numbers with as only operation S, the successor operation ('add one'). So the algebra we consider is:

$$N = \langle \mathbb{N}, S\rangle$$

where $S: \mathbb{N} \to \mathbb{N}$ is defined as 'addition with one'. We extend N with the operator $\oplus$, defined by the equalities $n \oplus 0 = n$ and $n \oplus S(m) = S(n\oplus m)$. This means that $\oplus$ is the usual addition operator. This operator is not polynomially definable over N. One sees this as follows. All polynomial symbols over N are of the form $S(S(\ldots S(x))$. So a polynomial symbol which corresponds with an operator which takes two arguments, contains only one variable, and is therefore dependent on only one of its arguments. Consequently the two place operation of addition cannot be defined polynomially.

In spite of the fact that $\oplus$ is not a polynomially definable operator, a (variant of) theorem 7.3 holds. For every algebra M and every $h \in Epi(N,M)$ there is an unique M' such that $h \in Epi(AddOp_{\oplus}N,M')$. This is proved as follows. Let $S^n(0)$ denote the n-times repeated application of S to 0; so $S^n(0) = S(S(\ldots S(0)..))$. For all $n \in N$ we have $n = S^n(0)$. Since $h \in Epi(N,M)$ this means that for all $m \in M$ there is an n such that $m = T^n(h(0))$, where T is the operator in M which corresponds with S. We define an operator $*$ in M as follows:

Assume: $m_1 = T^{n_1}(h(0))$ and $m_2 = T^{n_2}(h(0))$. Then $m_1 * m_2 =_d T^{n_1+n_2}(h(0))$. This definition is independent of the choice of n_1 and n_2 as is shown as follows:

Suppose that $m_1 = T^{n_1}(h(0)) = T^{n_3}(h(0))$, and that

$m_2 = T^{n_2}(h(0)) = T^{n_4}(h(0))$.

Then

$$T^{n_3+n_4}(h(0)) = T^{n_3}(T^{n_4}(h(0))) = T^{n_3}(T^{n_2}(h(0))) =$$
$$T^{n_3+n_2}(h(0)) = T^{n_2+n_3}(h(0)) = T^{n_2+n_1}(h(0)).$$

This shows that the definition of $*$ is a correct definition.
Now, let M' be $\text{AddOp}_*(M)$. Then $h \in \text{Epi}(N',M')$ since it is a surjective mapping and

$$h(n_1 \oplus n_2) = h(S^{n_1}(0) \oplus S^{n_2}(0)) = h(S^{n_1+n_2}(0)) =$$
$$T^{n_1+n_2}(h(0)) = T^{n_1}(h(0)) * T^{n_2}(h(0)) = h(S^{n_1}(0)) * h(S^{n_2}(0)) =$$
$$= h(n_1) * h(n_2).$$

To prove the unicity of M', we only have to prove the unicity of this definition of $*$. Suppose that $h \in \text{Epi}(\text{Add Op}_\oplus(N), M'')$, where the operation corresponding with $\oplus$ in M'' is $\circ$.

$m_1 \circ m_2 = T^{n_1}(h(0)) \circ T^{n_2}(h(0)) = h(S^{n_1}(0)) \circ h(S^{n_2}(0)) = h(S^{n_1}(0) \oplus S^{n_2}(0))$

$= h(S^{n_1+n_2}(0)) = T^{n_1+n_2}(h(0)) = m_1 * m_2$.

7.5. END

The characterization of operators which are safe is still an open question. But for a class of algebras which is relevant for us, such a characterization can be given. In the sequel we will always work with a logic which has as syntax a free algebra with infinitely many generators (all variables and constants are generators). For such algebras all safe operators are polynomially definable. Note that in example 7.5, were there was no polynomial definition for $\oplus$, there is a single generator (viz.0). Results related to the above one are given in Van BENTHEM (1979b); the proof of the above result is given in appendix 1 of this book. The notion of a 'safe deriver' is, related to the notions 'enrichment' and 'derivor' used in the theory of abstract data types, e.g. in ADJ 1978.

8. MONTAGUE GRAMMAR

The notion 'Montague grammar' is often used to indicate a class of grammars which resembles the grammar used in Montague's most influential publication PTQ (MONTAGUE 1973). It is, however, not always made clear what is to be understood by 'resembling' in this context. There are a lot of proposals which deviate from PTQ in important respects. Some proposals have a rather different syntax, other use a different logic or different models. The definition of 'Montague grammar' should make clear, which proposals are covered by this notion and which not.

In my opinion the essentail feature of a Montague grammar consists in its algebraic structure. The most pure (and most simple) definition would be that a Montague grammar consists in an algebraic grammar and a homomorphic interpretation. One always uses, in practice, some formal (logical) language as auxiliary language, and the language of which one wishes to describe the meanings is translated into this formal language. Thus the meaning assignment is performed indirectly. The aspect of translating into an auxiliary language is, in my opinion, unavoidable for practical reasons, and I therefore wish to incorporate this aspect in the definition of a Montague grammar. This decision includes (by suitable interpretation) grammars in which the interpretation is given directly. The most important example of that kind of grammar is the grammar in 'English as a formal language' (MONTAGUE 1970a). For such grammars the name simple Montague grammar seems suitable. These considerations should explain the following definitions.

8.1. DEFINITION. A *simple Montague grammar* consists of
1. an algebraic grammar A
2. an algebra M similar to A
3. a homomorphism $h \in \mathrm{Hom}(A,M)$.

8.2. DEFINITION. A *Montague grammar* consists of
1. an algebraic grammar A (the 'syntactic algebra')
2. an algebraic grammar L (the 'logical algebra')
3. an algebra M similar to L (the 'semantic algebra')
4. a homomorphism $h \in \mathrm{Hom}(L,M)$ (the 'interpretation of the logic')
5. an algebra D(L), similar to A derived from L, where D is a safe deriver.
6. a homomorphism $h \in \mathrm{Hom}(T_A, D(L))$ (the 'translation').

8.2 END

Definition 8.2 is illustrated by figure 2 (cf. figure 1.)

$$\begin{tikzcd}
 & T_A \arrow[d, "tr"] \\
L \arrow[r, dashed] \arrow[d, "h"] & D(L) \arrow[d, "h"] \\
M \arrow[r, dashed] & M'
\end{tikzcd}$$

Figure 2. A Montague grammar

The logical language which we will use is just as in PTQ, the language of intensional logic. Its (algebraic) grammar L and its (homomorphic) interpretation will be considered in chapter 3. The grammar A of the PTQ-fragment and its translation D(L) will be presented in chapter 4. The deriver D that will be used can be considered as being built from more elementary ones. I have found it convenient to define *four* more elementary derivers, but other decisions are possible as well. The most important deriver is AddOp which has been discussed in section 7. The other three are introduced below: first an informal discussion; then a formal definition.

The first deriver I will discuss is Add Sorts. An application has the form AddSorts $_{T,\sigma}(A)$, where T is a collection of sorts, and $\sigma\colon T \to S_A$ a function. The effect of this deriver is that a new algebra is formed with as sorts $T \cup S_A$, and with as carrier for $\tau \in T$ the set $A_{\sigma(\tau)}$. So this deriver introduces new sorts without introducing new elements. This deriver will be used when we need to introduce several new sorts which get their elements from one single old sort. An example of this as follows. In a syntax for English, nouns like *man* and verbs like *run* will be in different categories because they have different syntactic properties. But semantically both expressions are considered as predicates of the same type. Therefore we have to build from one old carrier (of predicates) two carriers (of nouns and of verbs). We may remove from each of these two carriers the elements which are not necessary for the translation of English, but in principle the carriers may still be non-disjoint (e.g. both may contain variables for predicates).

The second deriver I will discuss is DelOp. An application of this deriver has the form $\mathrm{DelOp}_\Delta(A)$. Here is Δ a subset of the set of operator symbols of A. The effect of this deriver is that a new algebra is formed which differs from A in the respect that it does not have the operators mentioned in Δ. This deriver is needed for the following reason. The derived algebra D(L), see figure 2, should only have operators which correspond with operators in A. Not all operators of the logical algebra L will be operators of D(L). For instance, the introduction of the universal quantifier might not correspond to any of the operations of the grammar A for the natural or programming language under consideration. Therefore we need a deriver which removes operators.

The last deriver is SubAlg. An application of this deriver has the form $\mathrm{SubAlg}_H(A)$. Here is H a sorted collection of elements of A. Its effect is that an algebra is formed which has the same operators as A, but which has H as a generating set. This deriver is used in the following kind of situation. The logical algebra L (see figure 2) has for each sort infinitely many generators. The grammar A might not have this property. For instance, the sentences of a natural language are all built up from smaller components, and hence there are no generators of the sort 'sentence'. SubAlg is then used to reduce the carriers of L to those elements which will be images of elements in A.

Below these three methods are defined, and their safeness is proven. It is not surprising that these methods are safe. Nevertheless the proofs are not elegant, but rather ad-hoc. This is probably due to the fact that there is hardly any theory about derivers of many-sorted algebras which I could use here. GOGUEN & BURSTALL (1978) present some category-theoretic considerations about derivers for many-sorted algebras in the sense of ADJ 1977. I have already mentioned the work of Van BENTHEM (1979b) concerning the introduction of new operators in one sorted algebras. That there is a need for a general theory appears, apart from the present context, in work in the field of abstract data types in the theory of programming languages, see e.g. EHRIG, KREOWSKI & PADAWITZ 1978 and ADJ 1978.

8.3. <u>DEFINITION</u>. Let $\sigma\colon T \to S$ arbitrary. Then $\sigma' = \bigcup_n (T\cup S)^n \times (T\cup S) \to \bigcup_n S^n \times S$ is defined by

1. $\sigma'(s) = s$ for $s \in S$
2. $\sigma'(t) = \sigma(t)$ for $t \in T$
3. $\sigma'(\langle\langle t_1,\ldots,t_n\rangle t_{n+1}\rangle) = \langle\langle\sigma'(t_1),\ldots,\sigma'(t_n)\rangle,\sigma'(t_{n+1})\rangle$.

In the sequel we will write σ for σ'.

8.4. THEOREM. *Let* $A = \langle(A_s)_{s\in S},(F_\gamma)_{\gamma\in\Gamma}\rangle$ *be a Σ-algebra. Let* T *be some set* ($S\cap T=\emptyset$) *and* σ: $T \to S$ *some mapping. Then there is an algebra*
$A' = \langle(A'_t)_{t\in T\cup S},G\rangle$ *where*
I. $A'_t = A_{\sigma'(t)}$
II. *The set of operators* G *is defined as follows: for all* $\gamma \in \Gamma$ *and all* $\langle w,u\rangle \in (T\cup S)^n\times(T\cup S)$ *with* $\sigma(\langle w,u\rangle) = \tau(\gamma)$ *we add a new operator* $g_{\gamma,\langle w,u\rangle}$ *of type* $\langle w,u\rangle$, *and define the effect of* $g_{\gamma,\langle w,u\rangle}$ *to be equal to* F_γ.
(So the operator indices are the compound symbols $\gamma,\langle w,u\rangle$*).*

PROOF. The elements of A and A' are equal. Since A is closed under F_γ, we have that A' is closed under $g_{\gamma,\langle w,u\rangle}$.

8.5. DEFINITION. The algebra introduced in 8.4 is denoted Add $\text{Sorts}_{\sigma,T}(A)$.

8.6. THEOREM. AddSorts *is safe.*

PROOF. Let $A = \langle(A_s)_{s\in S},(F_\gamma)_{\gamma\in\Gamma}\rangle$ and $B = \langle(B_s)_{s\in S},(G_\gamma)_{\gamma\in\Gamma}\rangle$ be similar algebras, and let $h \in \text{Epi}(A,B)$. Suppose σ: $T \to S$. Define $A' = \text{Add Sorts}_{\sigma,T}(A)$ and $B' = \text{Add Sorts}_{\sigma,T}(B)$. We now prove that $h \in \text{Epi}(A',B')$, and that B' is (up to isomorphism) the unique algebra with this property.

Since the elements in B' are the same as in B, the mapping h is surjective. Remains to show that h is a homomorphism. In analogy of theorem 8.4 we denote the operators introduced in $\text{AddSorts}_{\sigma,T}(B)$ by $g_{\gamma,\langle w,u\rangle,B}$, and those introduced in $\text{AddSorts}_{\sigma,T}(A)$ by $g_{\gamma,\langle w,u\rangle,A}$.
Then

$$h(g_{\gamma,\langle w,u\rangle,A}(a_1,\ldots,a_n)) = h(F_\gamma(a_1,\ldots,a_n)) =$$
$$= G_\gamma(h(a_1),\ldots,h(a_n)) = g_{\gamma,\langle w,u\rangle,B}(h(a_1),\ldots,h(a_n)).$$

That B' is unique can be proved in the same way as we did in the proof of 7.3 (if there was another algebra, it should have the same carriers and operations).

8.7. THEOREM. *Let* $A = <(A_s)_{s\in S}, \underline{F}>$ *be an algebra and let* $\Delta \subset \underline{F}$. *Then* $A = <(A_s)_{s\in S}, \underline{F}\backslash\Delta>$ *is an algebra.*

PROOF. A is closed under all F_γ from $\underline{F}\backslash\Delta$.

8.8. DEFINITION. The algebra introduced in 8.7 is denoted

$$\mathrm{DelOp}_\Delta(A).$$

8.9. THEOREM. DelOp *is safe.*

PROOF. Let A and B be similar algebras and $h \in \mathrm{Epi}(A,B)$.
Define $A' = \mathrm{DelOp}_\Delta(A)$ and $B' = \mathrm{DelOp}_\Delta(B)$. We now prove that $h \in \mathrm{Epi}(A',B')$ and that B' is the unique algebra with this property.
Algebras A' and B' are similar since if F_γ is an operation of A' then G_γ is an operation of B'. That h is surjective on B' is evident. Since h respects all F_γ from $\underline{F}$ it also does for those in $\underline{F}\backslash\Delta$. That B' is unique is proved in the same way as in 7.3.

8.10. THEOREM. *Let* $A = <(A_s)_{s\in S}, (F_\gamma)_{\gamma\in\Gamma}>$ *be an algebra and* $H = (H_s)_{s\in S}$ *a collection sets with* $H_s \subset A_s$. *Let* $B = <[(H_s)_{s\in S}], (F_\gamma)_{\gamma\in\Gamma}>$. *Define* $T = \{s \mid s \in S$ *such that* $H_s \neq \emptyset\}$. *Then* $B' = <(B_t)_{t\in T}, (F_\gamma)_{\gamma\in\Gamma}>$ *is an algebra.*

PROOF. $(B_t)_{t\in T}$ is closed under F_γ.

8.11. DEFINITION. The algebra B' from theorem 8.10 is denoted $\mathrm{SubAlg}_H(A)$.

8.12. THEOREM. SubAlg *is safe.*

PROOF. Let $A = <(A_s)_{s\in S}, (F_\gamma)_{\gamma\in\Gamma}>$ and $B = <(B_s)_{s\in S}, (G_\gamma)_{\gamma\in\Gamma}>$ be similar algebras and $h \in \mathrm{Epi}(A,B)$. Suppose that $H = (H_s)_{s\in S}$ is a collection such that $H_s \subset A_s$. Define $A' = \mathrm{SubAlg}_H(A)$ and $B' = \mathrm{SubAlg}_{h(H)}(B)$. We now prove that for $\hat{h} = h{\restriction}A'$ holds that $\hat{h} \in \mathrm{Epi}(A',B')$, and that B' is the unique algebra with this property.
First we proof that $\hat{h}(A'_s) = B'_s$.

I. $D = <((\hat{h}(A'_s))_{s\in S}, (G_\gamma)_{\gamma\in\Gamma}>$ is an algebra, see theorem 6.3. Since $H_s \subset A_s$ we have $\hat{h}(H_s) \subset h(A_s)$. So the generators of B' are in D, so $(B'_s) \subset \hat{h}(A'_s)$.

II. That $\hat{h}(A'_s) \subset (B'_s)$ is proved by induction:

First note that this is true for the generators of A'_s.
Suppose $\tau(\gamma) = <<s_1,\ldots,s_n>, s_{n+1}>$. Let $a_1 \in A'_{s_1},\ldots,a_n \in A'_{s_n}$, and assume that

$\hat{h}(a_1) \in B'_{s_1},\ldots,\hat{h}(a_n) \in B'_{s_n}$. Then $\hat{h}(F_\gamma(a_1,\ldots,a_n)) = \hat{G}_\gamma(\hat{h}(a_1),\ldots,\hat{h}(a_n))$ because $h \in Epi(A,B)$. Since B is an algebra we have $\hat{h}(F_\gamma(a_1,\ldots,a_n)) \in B'_{s_{n+1}}$.

From I and II it follows that $B' = \hat{h}(A'_s)$, hence $h \in Epi(A',B')$. That B' is unique can be proved in the same way as in 7.3.

8.1. END

Derivers like the ones defined above are not the only safe derivers. Taking a cartesian product or taking a projection from such a product are probably safe in some sense. Such derivers could be relevant for linguistic purposes. In the treatment of presuppositions (by KARTTUNEN & PETERS 1979) a phrase is connected with two formulas: one denoting its meaning, and one denoting its presuppositions. If two phrases are combined in the syntax to form a new phrase, then the two meanings are combined to form the meaning of the new phrase, and the presuppositions are combined to form a new presupposition. This situation fits into the framework if the new semantic algebra would be considered as the product of two copies of the same semantic algebra (however, details of their proposal give rise to complications).

The derivers described in this section together with AddOp are the only derivers we will use. They constitute the basis for the way in which I will introduce derived algebras. A derived algebra will be defined by providing in some way the following information

0) what the old algebra is
1) what the sorts of the new algebra are
2) what the generators of the new algebra are
3) what the operators of the new algebra are.

This information can be used in several ways to build a derived algebra. One might first add the new sorts and then the new operators, or vice versa. One might use the derivers described above, or variants of them. But all methods yield the same algebra, as follows from the uniqueness proof given in the next theorem.

8.13. THEOREM. *Let* $A = \langle(A_s)_{s\in S},(F_\gamma)_{\gamma\in\Gamma}\rangle$ *and* $B = \langle(B_s)_{s\in S},(G_\gamma)_{\gamma\in\Gamma}\rangle$ *be similar algebras and let* $h \in Epi(A,B)$. *Let furthermore be given*

1) *a collection of sorts* T *and a mapping* $\sigma\colon T \to S$,
2) *a collection of new generators* $(H_t)_{t\in T}$, *where* $H_t \subset A_{\sigma(t)}$
3) *a family of polynomials* $P = (p_i)_{i\in I}$ *and a type giving function* $f\colon (p_i)_{i\in I} \to \bigcup_n T^n \times T$ *such that* $f(p_i) = \langle w,t\rangle$ *only if* $p_i \in POL^A_{\langle\sigma(w),\sigma(t)\rangle}$.

Then there is a unique algebra $D = <(D_t)_{t\in T}, \Pi>$ *where*

1) $D_t \subset A_{\sigma(t)}$, *i.e. the carriers of* D *are subsets of the carriers of* A
2) $\Pi = \{p_{i,<w,v>,D} \mid p_i \in (P_i)_{i\in I},\ f(p_i) = <w,v>$ *and*

   $p_{i,<w,v>,D}(d_1,\ldots,d_n) = p_A(d_1,\ldots,d_n)\}$
3) D *is generated by the collection* $(H_t)_{t\in T}$, *i.e.* $D = <[(H_t)_{t\in T}], \Pi>$

Moreover, there is a unique algebra E *such that for* $\hat{h} = h \restriction D$ *we have* $\hat{h} \in \mathrm{Epi}(D,E)$.

<u>PROOF</u> I) *Existence of* D

The derived algebra will be defined in four steps which are indicated in figure 3.

$$\begin{tikzcd}
A \arrow[r, dashed] \arrow[d] & A_1 \arrow[r, dashed] \arrow[d] & A_2 \arrow[r, dashed] \arrow[d] & A_3 \arrow[r, dashed] \arrow[d] & A_4 = D \arrow[d] \\
B \arrow[r, dashed] & B_1 \arrow[r, dashed] & B_2 \arrow[r, dashed] & B_3 \arrow[r, dashed] & B_4 = E
\end{tikzcd}$$

<u>Figure 3.</u> Construction of D

The algebras mentioned in figure 3 are obtained using the derivers defined before.

$A_1 = \mathrm{AddOp}_P(A)$,	so A_1 is obtained by adding all polynomial symbols mentioned in P.
$A_2 = \mathrm{AddSorts}_{T,\sigma}(A_1)$	so A_2 is obtained from A_1 by adding the sorts of T
$A_3 = \mathrm{DelOp}_\Delta(A_2)$	where $\Delta = \{\delta \mid f$ is an operator symbol of A_2 and $\delta \notin \Pi\}$ so A_3 has only operators symbols given in P with the type indicated by f.
$A_4 = \mathrm{SubAlg}_{(H_t)_{t\in T}}(A_3)$	so A_4 is the algebra built from $(H_t)_{t\in T}$.

From the previous definitions concerning derivers it follows that A_4 is an algebra which satisfies the three conditions mentioned in the theorem.

II. *Uniqueness of* D

Suppose that D1 and D2 are algebras satisfying the requirements for D. Define $D3_t = D1_t \cap D2_t$. Then $D3 = <(D3_t)_{t\in T}, \Pi>$ is a subalgebra of D1 because

1) $D3_t \subset D1_t$
2) $(D3_t)_{t\in T}$ is closed under the operations in Π:
Let the type of $\pi \in \Pi$ be $<<t_1,\ldots,t_n>,t_{n+1}>$ and suppose $d_1 \in D3_{t_1},\ldots,d_n \in D3_{t_n}$. Then $\pi(d_1,\ldots,d_n) \in (D1_{t_{n+1}} \cap D2_{t_{n+1}})$ since π has the same interpretation in D1 and D2, and both D1 and D2 are closed under π. Hence D3 is closed under π.

Moreover, $H_t \subset D3_t$. So D3 is a subalgebra of D1 containing the generators of D1, hence D3 = D1. From this follows that D2 = D1.

III. *Existence and unicity of* E

Algebra E is defined by analogy to the definition of D. So $B_1 = AddOp_{h(P)}(B)$, $B2 = AddSorts_{T,\sigma}(B1)$, $B3 = DelOp_\Delta(B_2)$ and $B4 = SubAlg_{(H_t)_{t\in T}}(B3)$. Here h(P) is defined as in theorem 7.3 and Δ is defined as above. From the previous theorems about derivers it follows that $h \in Epi(A_i,B_i)$ for $i \in \{1,2,3\}$ and that $h\restriction A4 \in Epi(A4,B4)$. It also follows that each B_i is the unique algebra with this property. In particular B4 is the unique algebra which satisfies the requirement for E.

8.14. <u>EXAMPLE</u>. This example consists of the syntax and semantics of a small fragment of English. The meanings of the sentences of the fragment are obtained by translating them into an algebra derived from predicate logic. Its semantics is very primitive: the meaning of a sentence is a truth value. This aspect is not important because the purpose of the example is to illustrate the derivers described in this section.

The generators of the algebraic syntax are as follows:

$B_T = \{John, Mary\}$

$B_{IV} = \{run, walk\}$

$B_{CN} = \{child, professor\}$.

The rules of the syntax are as follows

F_1: T × IV → S defined by $F_1(\alpha,\beta) = \alpha\ \beta s$.

F_2: T × CN → S defined by $F_2(\alpha,\beta) = \alpha$ *is a* β.

Examples are:

F_1(*John,run*) = *John runs*

F_2(*Mary,professor*) = *Mary is a professor.*

This information determines the following algebraic grammar

$<<[\{John, Mary\}_T, \{run, walk\}_{IV}, \{child, professor\}_{CN}], \{F_1, F_2\}>, S>$.

The fragment is translated into a derived algebra which is determined by the following information.
The elements of B_T are translated respectively into *john,mary* which are constants of type e. The elements of B_{IV} and B_{CN} are translated into the following constants of type <e,t>: *run,walk,child,professor.* Notice the different type face used for English words and logical constants. The application of operator T_1 (corresponding to rule F_1) is described by the polynomial symbol $x_{2,<e,t>}(x_{1,e})$. Consequently, if α' and β' are the translations of α and β respectively, then the translation of the term $F_1(\alpha,\beta)$ is $\beta'(\alpha')$. The operator T_2 (corresponding with rule F_2) is defined by the same polynomial symbol. Examples are:

translation of F_1(*John,run*) is *run(john)*.

translation of F_2(*Mary,professor*) is *professor(mary)*.

The description of the derived algebra given above has not the form used in theorem 8.12. But implicitly all that information is given, as appears from the following.

1. The sorts of the derived algebra are the same as those of the syntax: T,IV,CN and S. The mapping σ to the old sorts is σ(T) = e, σ(IV) = <e,t>, σ(CN) = <e,t> and σ(S) = t.
2. The generators of the derived algebra are the translations of the generators of the syntactic algebra.
3. The polynomial operator is $x_{2,<e,t>}(x_{1,t})$, and the type-giving function f says $f((x_{2,<e,t>}(x_{1,t})) = \{<<T,IV>,S>,<<T,CN>,S>\}$.

The process of making a derived algebra as is described in the proof of theorem 8.12 proceeds as follows.
step 1. The polynomial operator $x_{2,<e,t>}(x_{1,t})$ is added to the algebra of predicate logic.

step 2. The categories T,CN,IV and S are added. Their carriers consist of the expressions of type e, type <e,t>, type <e,t> and type t respectively. Moreover, the operators are multiplied. For instance, the operator $x_{2,<e,t>}(x_{1,e})$ gets 12 incarnations. Examples of the types of these incarnations are <<s,e>,e>,t>, <<CN,T>S>, <<s,e>,T>S> and <IV,T>t>.
step 3. Everything that is not needed will be removed. The carriers of sorts CN and IV are reduced, which has the effect that they become disjunct. Sort t is removed, and most incarnations of the polynomial are removed, except for <<CN,T>S> and <<IV,T>,S>.

The derived algebra which results from this process is the unique algebra guaranteed in theorem 8.13. In the sequel I will present the information needed to apply theorem 8.11 in the implicit way used here. The process of forming a derived algebra will not be described explicitly.

9. DISCUSSION

The framework defined in this paper is closely related to two proposals in the literature. These proposals are developed in two quite different fields of semantics. The first one is developed by Richard Montague for the treatment of the semantics of natural languages. It is presented in "Universal Grammar" (MONTAGUE 1970b), henceforth UG. The first sentence of this article reads

> *There is in my opinion no important theoretical difference between natural languages and the artificial languages of logicians; indeed, I consider it possible to comprehend the syntax and semantics of both kinds of languages within a single natural and mathematical precise theory.*

It is striking to discover that this statement also holds for the languages of computer scientists. Independent of Montague's work, and independent of the philosophical tradition this work was based on, the same ideas were developed in the field of semantics of programming languages by the group called Adj (Goguen, Thatcher, Wagner, Wright). Their motivation had nothing to do with the compositionality principle; they have a practical justification for their framework. The second sentence of ADJ 1979 reads:

> *The belief that the ideas presented here are key, comes from our experience over the last eight years in developing and applying these concepts.*

A more detailed comparision between these proposals and the one described in this chapter will be given below.

The basic difference between Montague's framework and the present one, is that Montague did not have the notion 'many sorted algebra' available. He worked with a one sorted algebra and his syntax consisted of the description of a very special one-sorted algebra: one with much additional structure. I have a much more general algebraic concept of syntax and his one-sorted algebra is a special case. However, the mathematical object Montague defines is the same as the object I define. The two frameworks present different views of the same mathematical object. These different views have some consequences for the details of the framework.

1. In the present framework operators are typed. In Montague's framework operators are typeless, but rules are typed. This has the following consequence. If we apply the UG framework to PTQ, then not the syntactic rules (i.e. S4,S5,...) are the operators in the algebraic sense, but the operations on strings (i.e. F_1,F_2...). The framework requires for each F a single corresponding semantic operation. But this is not for all F's the case (e.g. not for F8: conjunction-operation). This illustrates that the present framework, in which rules and operators coincide, gives an approach which is closer to practice than the UG framework.

2. Both frameworks require that the operators be total. In my framework this means that an operator has to be defined for the whole carrier of the type of its arguments. In Montague's framework it means that the operators have to be defined for all elements on the algebra, even for those elements to which it will never be applied. A similar remark holds for homomorphism. For instance the semantic interpretation has to be defined for expressions which are not expressions of logic such as $\rightarrow \rightarrow p$. In practice no one actually defines homomorphisms for such arguments. In the present framework this practice is sanctioned.

3. In the present framework, there is a natural relation between the disambiguated and the generated language: from an expression in the term algebra one obtains the corresponding expression in the generated language by evaluating the expression. In UG there is a (further unspecified) relation R relating the disambiguated language with the generated language. Such a relation can be used for several purposes: for neat ones such as deleting brackets, but also for filtering, completely reformulating the expression, or building new structures and other obscure operations. That R can be any such relation is not good. As far as I know, no one working in Montague grammar actually uses this extreme power of R. Hence it is attractive to restrict R as we have done.

4. The present framework has some built in restrictions to guarantee that the grammar be effective. The restrictions are obeyed by all existing proposals. The original, unrestricted definitions allow for too uninteresting grammars.

Summarizing, the differences between the present framework and Montague's have as a consequence that the present framework is much closer to practice, and that unwanted, and unused, facilities are no longer available.

Next I will consider the relation of our framework to that of Adj. The basic idea underlying their approach is formulated in ADJ (1977, p.69).

> *In the cases we examine, syntax is an initial algebra, and any other algebra* A *in the class is a* ***possible domain*** *(or* ***semantic algebra****); the* ***semantic function*** *is the uniquely determined homomorphism* h_A: S → A, *assigning a* ***meaning*** h_A(s) *in* A *to each syntactic structure* s *in* S.

This statement implies that the group Adj works with what we have called 'simple Montague grammars'. They have, however, not explicitly described the framework in which they work. They are interested primarily in practical work concerning the semantics of programming languages. It appears that their work is in accordance with what we have defined as being a (standard) Montague grammar. For instance, a central aspect of the present framework is that polynomial operators are used to define complex operations on meanings. Adj certainly knew about the benefit of polynomials: their papers are full of such operators. But no explicit formulation is given of the role of polynomials in their approach. Since a framework is only implicit, it is possible, that the algebraic theory developed in this chapter is hidden in their works. The most fundamental difference between the two approaches is that they base the semantics on the algebraic grammar for the language, whereas we base it on the corresponding term algebra (i.e. on derivational histories). However, since the framework of Adj is not made explicit, it is difficult to compare their approach with ours. Therefore I restrict myself to the general remarks given above. Below I will discuss some technical differences in the definition of algebra and homomorphism.

In the Adj approach it is required that all similar algebras have the same operator symbols. I prefer to have the possibility of using different operator symbols because that is standard in the field of Montague grammars (the operators from the syntactic algebra are usually denoted S_i, and those of the logical algebra T_i). Furthermore, the Adj definition has as a consequence that renaming the sorts gives rise to a completely new algebra:

an algebra obtained by renaming the sorts is not isomorphic to the original algebra, it is not even similar! For these reasons I prefer the more general definition of many sorted algebra and of similar algebras which are used in this chapter.

In the theory of universal algebras one usually allows for nullary operations, i.e. for operations which do not take an argument and which always yield the same value. In our definition such operations are not allowed. To consider constants as nullary operators is intuitively difficult, and practically inconvenient. For, instance, after having presented their definition, which allows for nullary operations, ADJ (1977,p.71) says that the uniformity is 'mathematically nice', but 'it is often more convenient' to separate them out from the more general operators. Another difficulty is the following. Let an algebraic grammar be given for a certain fragment. Suppose that a new element is added to an existing carrier (a new word is added of an already present category). Then one would judge intuitively that nothing essential is added. If a new rule is added (i.e. a non-nullary operation), then a new type of syntactic constructions is added to the fragment. In such a situation one would say that something essentially is added. If nullary operations would be allowed for, then these two kinds of addition would have the same status, which is not in accordance with practice. However, this difference concerning operators does not give rise to essential differences in the algebraic theory (e.g. because I have adapted suitably the definition of 'polynomial symbol').

In our approach a homomorphism is a mapping with as domain the elements of the carriers. In the Adj approach it is a sorted collection of mappings. For each sort there is a separate mapping. So in case an element occurs in several carriers it is treated as if there are two different elements. The images under the homomorphism can be different for the same element in different sorts. This is not acceptable in our approach. In the process of making a derived algebra we impose a new structure of sorts on the logical algebra, and the interpretation homomorphism has to determine uniquely the interpretation of the elements of the new sorts (which are also elements of the old sorts). If the homomorphism is defined as a sorted collection of functions, the interpretation of the new carriers is arbitrary. Hence theorem 8.12 would not be valid. In order to guarantee an unique interpretation for the derived algebra, the Adj-definition was corrected.

Summarising, the main difference between our approach and that of Adj is that we base the semantics on the term algebra, whereas Adj does not. Another difference is that we have an explicit framework. Differences in technical details are a consequence of this framework or of requirements from established practice. The present framework might be considered as a synthesis of the idea's of Montague with technical tools of Adj.

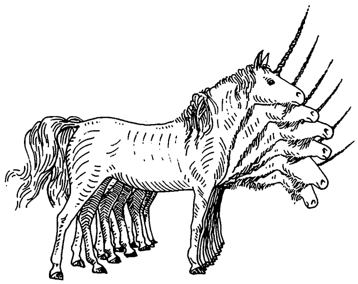
Finally, I will mention some afterthoughts about two points made in this chapter. The choice not to allow for nullary operators is non-standard and has some advantages for the explication. Along this way we came just far enough. But this choice has the disadvantage that existing theory cannot be applied directly. More in particular, I did not succeed in obtaining a handsome definition of a 'free algebra'. Maybe this is a sign that it might be wiser to follow the standard definition and accept the didactic difficulties. The second point concerns the discussion in section 5(p.67) of some remarks of ADJ concerning the context-freeness of the generated language. Joe Goguen (pers.comm) explained that ADJ's remarks should not be understood in literal way. Ind ADJ 1977 the possibility is mentioned that the set of operators is infinite. In that way non-context free languages can be dealt with. The criticism on their approach should therefore not be that ADJ can only deal with context-free languages, but that they can only deal with grammars with context free rules, but not with arbitrary syntactic operations.

CHAPTER III

INTENSIONAL LOGIC

ABSTRACT

In this chapter the language of intensional logic is introduced; this language is a useful tool for representing meanings of e.g. English. The semantic interpretation of intensional logic is defined by a translation into the language Ty2 of two-sorted type theory. Several properties of intensional logic are explained using this translation into Ty2.



1. TWO FACETS

1.1. Introduction

Our aim is to associate in a systematic way the expressions of a language with their meanings. Hence we need a method to represent meanings. The most convenient way to do so, is to use some suitable logical language. Once the interpretation of that language has been defined, it can further be used to represent meanings. The language we will use in this book, is the language of intensional logic, henceforth IL. This language is especially suitable for representing the intended meanings because it 'wears its interpretation upon its sleeves' (Van Benthem, pers.comm.).

In chapter 1 some consequences of the principle of compositionality are discussed. Here I will pay special attention to two of them.

I) The meanings associated with expressions of a natural language or a programming language are intensions, i.e. functions on a domain consisting of a set of 'indices'. The indices formalize several factors which influence the meaning of an expression.

II) The meanings form a many sorted algebra which is similar to the syntactic algebra. Hence we have for each category in the syntactic algebra a corresponding sort in the semantic algebra: the semantic model is 'typed'.

In the light of the close connection between IL and its models, it is not surprising that these two facets of meaning are reflected in IL. This language contains operators connected with indices (e.g. tense operators), as well as operators reflecting the typed structure of the semantic domain (e.g. λ abstraction). This means that IL can be considered as the *amalgamation* of two kinds of languages: type logic and modal tense logic. With this characterization in mind, many properties of IL can be explained. This will be done in the sequel.

1.2. Model - part I

The set of indices plays an important role in the formalization of the notion 'meaning' since meanings are functions with indices as their domain. The definition of the model will not say much about what indices are: they are defined as an arbitrary set. This level of abstraction has the advantage that the meanings of such different languages as English and Algol can be described by them. But one might like to have an intuitive understanding of what indices are, what they are a formal counterpart of, and which degree of reality they have. Several views on these issues are possible, and I will mention some of them. Thereafter I will give my personal opinion.

1. Our semantic theory gives a model of how the reality is, or might have been. An index represents one of these possibilities. In application to natural language this means that an index represents a possible state of affairs of the reality. In application to programming languages this means that an index represents a possible internal state of the computer.
2. Our semantical theory gives a model of a psychologically acceptable way of dealing with meanings. In this conception an index formalizes a perceptually possible state of affairs (cf. PARTEE 1977b).
3. Languages describe concepts, and users of a language are equipped with a battery of identification procedures for such concepts. An index represents a class of possible outcomes of such procedures (cf. TICHY 1971).
4. Our semantic theory describes how we deal with data. An index represents a maximal, non-contradictory set of data (cf. VELTMAN 1981).
5. 'In order to say what meaning is, we may first ask what a meaning does, and then find something that does that.' (LEWIS 1970). We want meanings to do certain things (e.g. formalize implication relations among sentences), we define meanings in an appropriate way, and indices form a technical tool which is useful to this purpose. Indices are not a formalization of something; they are just a tool.

Conception 1 is intuitively very appealing, and most widespread in the literature. But the interpretations 2,3, and 4 are also intuitively appealing. The reader is invited to choose that conception he likes best. An intuitively conceivable interpretation might help him to understand how and why everything works. But the reader should only stick to his interpretation as long as it is of use to him. For the simple cases indices can probably be considered as an adequate formalization of his intuitions. But once comes the day that his intuition does not help him any more. Then he should switch to conception 5; no interpretation but a technical tool. Such a situation arises, for instance, with the treatment of questions. Do you have an idea of what the meaning of a question should be in the light of conception 1,2,3, or 4? For instance the treatment of indirect questions given in GROENENDIJK & STOKHOF (1981) cannot be explained on the basis of the first four conceptions. They have chosen as meanings those semantic entities which do what they wanted them to do: the indices play just a technical role.

1.3. Model - part II

In the model theory of type-logic the models are constructed from a few basic sets by adding sets of functions between already available sets. Two kinds of models can be distinguished, depending on how many functions are added. In the so called 'standard models', the addition clause says that if A and B are sets in the model, then the set A^B of all functions from B to A also is a set in the model. In the so called 'generalized models' one needs not to take this whole set, but one may take some subset. There is a condition on the construction of models which guarantees that not too few elements are added to the model: every object that can be described in the logic should be incorporated in the model.

The laws of type logic which hold in standard models are not axiomatizable. In order to escape this situation, the generalized models were introduced (HENKIN 1950). By extending the class of possible models, the laws were restricted to an axiomatizable class: the more models the more possible counter examples, and therefore the fewer laws.

What kind of models will be used for the interpretation of intensional logic? I mention four options.

1. the class of all standard models
2. a subclass of the standard models
3. the class of all generalized models
4. a subclass of the generalized models.

Which choice is made, depends on the application one has in mind, and what conception one has about the role of the model (see section 1.2). If one intends to model certain psychological insights, then one might argue that the generalized models with countably many elements are the best choice (cf. PARTEE 1977b, and the discussion in chapter 7). If the model is used for dealing with the semantics of programming languages then a certain subset of the generalized models is required (see chapter 4). In the application of Montague grammar to natural language, one works with option 2. A subclass of the standard models is characterized by means of meaning postulates which give restrictions on the interpretation of the constants of the logic. I will follow this standard approach in the sequel.

1.4. Laws

Most of the proof-theoretic properties of IL can be explained by considering IL as the union of two systems: type logic and modal tense logic. The modal laws of IL are the laws of the modal logic S5. Many of the laws of type logic are laws of IL, exceptions are variants of the laws which are not valid in modal logic. The laws of type logic (i.e. those which hold in all standard models) are not axiomatizable. Since IL has (on sorted) type logic as a sublanguage, IL is not axiomatizable either. For modal logic there is an axiomatization of the laws which hold in all generalized models. This is expressed by saying that type logic has the property of generalized completeness. By combining these two completeness results, the generalized completeness of IL can be proved (see also section 3).

1.5. Method

I have explained that many aspects of IL can be understood by considering IL as the amalgamation of type logic and modal tense logic. Nevertheless, the formal introduction of IL will not proceed along this line. I will first introduce some other language: Ty2, the language of two sorted type theory. On the basis of Ty2 I will define IL: the algebraic grammar of IL is an algebra derived from the algebraic grammar for Ty2. The reasons for preferring this approach are the following:

1. *Model theoretic*

In Ty2 the indices are treated as elements of a certain type just like all elements. This is not the case for IL. In the interpretation of IL indices occur only as domains of certain functions, but not as range. Therefore the models for IL become structures in which carriers of certain types are deleted, whereas, from the viewpoint of an elegant construction, these carriers should be there. In the models for Ty2 they are there. Remarkable properties of IL can be explained from the fact that these carriers are not incorporated in its models. It appears to be better for understanding, and technically more convenient, to describe first the full model, and to remove next certain sets, instead of to start immediately with the remarkable model.

2. *Homomorphisms*

From the viewpoint of our framework, it is essential to demonstrate that the interpretation of IL is a homomorphism. It seems, however, rather difficult to show that the interpretation homomorphism for type logic and that for modal tense logic can be combined to a single homomorphism for IL. Furthermore, we should, in such an approach, consider first the interpretations of these two languages separately. It is easier to consider only Ty2.

3. *Laws*
Many of the proof rules for Ty2 are easy to formulate and to understand. This is not the case with IL. It is for instance much easier to prove lambda conversion first for Ty2, and derive from this the rule for IL, than to prove the IL rule directly.

4. *Speculation*
We will use IL for expressing meanings of natural language expressions because it is a suitable language for that purpose. No explicit reference to indices is possible in IL, and there is no need to do so for the fragment we will consider. But one may expect that for larger fragments it is unavoidable to have in the logic explicit reference to indices. NEEDHAM (1975) has given philosophical arguments for this opinion, Van BENTHEM (1977) has given technical arguments, and GROENENDIJK & STOKHOF (1981) treat a fragment of natural language where the use of Ty2 turned out to be required. Furthermore we will consider in chapter 4 a kind of semantics for programming languages which requires that states can be mentioned explicitly in the logical language, and we will use Ty2 for that purpose.

2. TWO-SORTED TYPE THEORY

In this section the language Ty2 will be defined: the language of two sorted type theory. The name (due to GALLIN 1975) reflects that the language has two basic types (besides the type of truth values). It is a generalization of one sorted type theory which has (besides the type of truth values) one basic type. The language is defined here by means of an algebraic grammar.

Since in logic it is customary to speak of types, rather than of sorts, I will use this terminology, even in an algebraic context. The collection of types of Ty2 is the smallest set Ty such that
1. $\{e,s,t\} \subset Ty$ (e='entity',s='sense',t='truth value').
2. if $\sigma \in Ty$ and $\tau \in Ty$ then $<\sigma,\tau> \in Ty$.
This is the standard notation for types which is used in Montague grammar. It is, however, not the standard notation in type theory. Following CHURCH (1940), the standard notation is $(\sigma)\tau$ instead of $<\sigma,\tau>$. I agree with LINK & VARGA (1975) that if we would adopt that notation and some standard conventions from type theory, this would give rise to a simpler notation than

the one defined above. But I prefer not to confuse readers familiar with Montague grammar, and therefore I will use his notation.

For each type $\tau \in Ty$ we have two denumerable sets of symbols:

$$CON_{\tau} = \{c_{1,\tau}, c_{2,\tau}, \ldots\} \qquad \text{the constants of type } \tau$$

and

$$VAR_{\tau} = \{v_{1,\tau}, v_{2,\tau}, \ldots\} \qquad \text{the variables of type } \tau.$$

So the constants and variables are indexed by a natural number and a type symbol. The elements of VAR_{τ} are called variables since they will be used in Ty2 as variables in the logical sense (they should not be confused with variables in the algebraic sense which occur in polynomials over Ty2).

The generators of type τ are the variables and constants of type τ. The carrier of type τ is denoted as ME_{τ} (meaningful expression of type τ). An element of the algebra is called a (meaningful) expression. The standard convention is to call the meaningful expressions of type t 'formulas', but I will call all meaningful expressions 'formulas'. (this gives the possibility to distinguish them easily from expressions in other languages).

There are denumerable many operators in the algebra of Ty2, because the operators defined below are rather schemes of operators, in which the types involved occur as parameters. For instance the operator for equalities (i.e. $R_{=}$) corresponds with a whole class of operators: for each type $\tau \in Ty$ there is an operator $R_{=,\tau}$. These operators $R_{=,\tau}$ all have the same effect: $R_{=,\tau}(\alpha,\beta)$ is defined as being the expression $[\alpha=\beta]$. Therefore we can define a whole class with a single scheme. The scheme for $R_{=}$ should contain τ as a parameter, and other operations should contain two types as parameter. These types are not explicitly mentioned as parameter, since they can easily be derived from the context. The proliferation of operators just sketched is a consequence of the algebraic approach and caused by the fact that, if two expressions belong to different sorts for one operation (say for function application), they belong to different sorts for all operations (so for equality).

The operators of Ty2 are defined as follows

1. *Equality*

$$R_{=}: ME_{\tau} \times ME_{\tau} \rightarrow ME_{t} \quad \text{where } R_{=}(\alpha,\beta) = [\alpha=\beta].$$

2. *Function Application*

$$R_{(\,)}: ME_{<\sigma,\tau>} \times ME_{\sigma} \rightarrow ME_{\tau} \quad \text{where } R_{(\,)}(\alpha,\beta) = [\alpha(\beta)].$$

3. *Quantification*

$R_{\exists v}: ME_t \rightarrow ME_t$ where $v \in VAR_\tau$ and $R_{\exists v}(\phi) = \exists v[\phi]$.

For universal quantification ($R_{\forall v}$) analogously. Recall that in chapter 1 arguments were given for not analyzing $\exists v$ any further.

4. *Abstraction*

$R_{\lambda v}: ME_\tau \rightarrow ME_{<\sigma,\tau>}$ where $v \in VAR_\sigma$ and $R_{\lambda v}(\alpha) = \lambda v[\alpha]$.

5. *Connectives*

$R_\wedge: ME_t \times ME_t \rightarrow ME_t$ where $R_\wedge(\alpha,\beta) = [\alpha \wedge \beta]$.

Analogously for $R_\vee$, $R_\rightarrow$, and $R_\leftrightarrow$.

$R_\neg: ME_t \rightarrow ME_t$ where $R_\neg(\phi) = [\neg\phi]$.

The (syncategorematic) symbols [and] are used to guarantee unique readability of the formulas. They will be omitted when no confusion is likely. The syncategorematic symbols $\exists v$ (existential quantifier), $\forall v$ (universal quantifier) and λv (the lambda-abstraction) are called binders. A variable is called free when it does not occur within the scope of $\exists v$, $\forall v$, or λv. The notions 'scope' and 'free' can be defined rigorously in the usual way.

This completes the definition of the operators of Ty2. For the semantics of English, two more operators are needed. They introduce ordering symbols between expressions of type s (i.e. between index expressions).

6. *Ordering*

$R_<: ME_s \times ME_s \rightarrow ME_t$ $R_<(\alpha,\beta) = [\alpha < \beta]$

$R_>: ME_s \times ME_s \rightarrow ME_t$ $R_>(\alpha,\beta) = [\alpha > \beta]$.

After having described the sets of sorts, generators and operators of Ty2, I will present the algebraic grammar for Ty2. Let $\underline{R}$ be the collection of operators introduced in clauses 1-5. Then an algebraic grammar for Ty2 is

$$<< \left[(CON_\tau \cup VAR_\tau)_{\tau \in Ty} \right], \underline{R}>, t>.$$

If $\underline{R}$ is replaced by $\underline{R} \cup \{R_<, R_>\}$: then an algebraic grammar for Ty2< is obtained.

3. THE INTERPRETATION OF Ty2

The semantic domain in which Ty2 will be interpreted, consists of a large collection of sets, which are built from a few basic ones. These basic sets are the set A of entities, the set I of indices, and the set {0,1} of truth values. The sets D_τ of possible denotations of type τ are defined by;

1. $D_t = \{0,1\}, \quad D_e = A, \quad D_s = I$
2. $D_{<\sigma,\tau>} = D_\tau^{D_\sigma}$.

In order to deal with logical variables, we need functions which assign semantical objects to them. The collection AS of variable assignments (based on A and I) is defined by

$$AS = \prod_{\tau\in Ty} (D_\tau^{VAR_\tau}).$$

Let as, as' $\in$ AS. We say that as' $\underset{v}{\sim}$ as (as' is a v-variant of as) if for all $w \in$ VAR such that $w \not\equiv v$ holds that as(w) = as'(w). If as' $\underset{v}{\sim}$ as and as'(v) = d then we write $[v{\mapsto}d]$as for as'.

Now the necessary preparations are made to say what the elements of the semantic domains are. Let A and I be non-empty sets, and let D_τ be defined as above. The sets M_τ of meanings of type τ (based upon A and I) are:

$$M_\tau = D_\tau^{AS}.$$

By the semantic domain based upon A and I, we understand the collection $(M_\tau)_{\tau\in Ty}$. In such domains we will interpret Ty2. For Ty2< additional structure is required. The set I has to be the cartesian product of two sets W and T, where T is linearly ordered by a relation <. Here W is called the collection of possible worlds, and T the collection of time points. An element i $\in$ W $\times$ T is called a *reference point* or *index*.

As is suggested by the definition of semantic domain, the interpretation homomorphism of Ty2 will assign to an expression of type τ some element of M_τ, i.e. the meaning of $\phi \in ME_\tau$ is some function f: AS $\to D_\tau$. In chapter one we have formalized the meaning of an expression of predicate logic as a function f: AS $\to D_t$, and here this approach is generalized to other types. In the case of predicate logic a geometrical interpretation of this process was possible, and this led us towards the cylindric algebras. For the case of Ty2 it is not that easy to find a geometrical interpretation. In any case, I will not try to give one. But I consider the interpretation of Ty2 given here as a generalization of the interpretation of predicate logic with cylindric algebras. Therefore I will call the interpretation of quantifiers of Ty2 'cylindrifications'.

Analogous to the interpretation of variables, there are functions for the interpretation of constants. The collection F (based upon A and I) of functions interpreting constants is defined by:

$$F = \prod_{\tau \in Ty} D_\tau^{CON_\tau}.$$

By a model for Ty2 we understand a pair <M,F> where

1. M is a semantic domain (based on A and I).
2. $F \in F$, hence F is a function for the interpretation of constants (based on the same sets A and I).

In order to define a homomorphism from Ty2 to some model, the models should obtain the structure of an algebra similar to the syntactic algebra of Ty2. That means that I have to say what the carriers, the generators, and the operators of the models are. The generators and operators will be defined below, along with the definition of the interpretation homomorphism V (V = 'valuation'). The carrier of type τ has already been defined; viz. M_τ. So the value of an element of ME_τ under this interpretation V is a function from AS to D_τ. This function will be defined by saying what its value is for an arbitrary assignment as $\in$ AS. I will write $V_{as}(\alpha)$ instead of $V(\alpha)(as)$ because the former notation is the standard one.

The generators of the semantic algebra are the images of the generators of the syntactic algebra. These are defined by

a) $V_{as}(v_{\tau,n}) = as(v_{\tau,n})$

b) $V_{as}(c_{\tau,n}) = F(c_{\tau,n})$.

As for the last clause, one should remember that we are defining the interpretation with respect to some model, and that models are defined as consisting of a large collection of sets and a function F which interprets the constants.

The interpretation of compound expressions of Ty2 will be defined next. Let R be some operator of the syntactic algebra of Ty2. Then the value $V_{as}(R(\alpha))$ will be defined in terms of $V_{as}(\alpha)$. In this way it is determined how $V(R(\alpha))$ is obtained from $V(\alpha)$. Then it is also determined how the operator T, which produces $V(R(\alpha))$ out of $V(\alpha)$ is defined. For each clause in the definition of $V_{as}(\alpha)$, I will informally describe which semantic operator T is introduced.

1. *Equality*

$$V_{as}(\alpha=\beta) = \begin{cases} 1 & \text{if } V_{as}(\alpha) = V_{as}(\beta) \\ 0 & \text{otherwise.} \end{cases}$$

So $V(\alpha=\beta)$ is a function from assignments to truth values yielding 1 if $V(\alpha)$ and $V(\beta)$ get the same interpretation for that assignment. Consequently $T_=$ is the assignment-wise evaluated equality. To be completely correct, I have to say that there is a class of semantic operators described here and not a single one: for each type τ there is an equality operator $T_{=,\tau}$.

2. *Function application*

$$V_{as}(\alpha(\beta)) = V_{as}(\alpha)(V_{as}(\beta)).$$

So if $V(\alpha) \in M_{<\sigma,\tau>}$, $V(\beta) \in M_\sigma$, then $V(\alpha(\beta)) \in M_\tau$. And $T_{(\,)}: M_{<\sigma,\tau>} \times M_\sigma \to M_\tau$, where $T_{(\,)}$ is assignment-wise function application of the assignment-wise determined function to the assignment-wise determined argument.

3. *Quantification*

$$V_{as}(\exists v\phi) = \begin{cases} 1 & \text{if there is an } as' \underset{v}{\sim} as \text{ such that } V_{as'}(\phi) = 1. \\ 0 & \text{otherwise.} \end{cases}$$

The element $V(\exists v\phi) \in M_t$ is obtained from $V(\phi) \in M_t$ by application of $T_{\exists v}$. This operation $T_{\exists v}$ is a cylindrification operation like the ones introduced in chapter 1. $V_{as}(\forall v\phi)$ is defined analogously.

4. *Abstraction*

Let $v \in VAR_\sigma$ and $\phi \in ME_\tau$. Then $V_{as}(\lambda v\phi)$ is that function f with domain D_σ such that whenever d is in that domain, then f(d) is $V_{as'}(\phi)$, where $as' = [v\to d]as$. In the sequel I will symbolize this rather long phrase as

$$V_{as}(\lambda v\phi) = \underline{\lambda}d\ V_{[v\mapsto d]as}(\phi).$$

Here $\underline{\lambda}$ might be considered as an abstraction in the meta language, but its role is nothing more than abbreviating the phrase mentioned above: 'that function which ...'. The semantic operator $T_{\lambda v}$ corresponding to $R_{\lambda v}$ is a function from M_τ to $M_{<\sigma,\tau>}$ where $T_{\lambda v}$ associates with an element $e \in M_\tau$ some function $f \in M_{<\sigma,\tau>}$. This function f assigns to an as $\in$ AS the function that for argument d has the value e(as'), where as' = $[v \mapsto d]$as.

5. *Connectives*

$$V_{as}(\phi\wedge\psi) = \begin{cases} 1 & \text{if } V_{as}(\phi) = V_{as}(\psi) = 1 \\ 0 & \text{otherwise.} \end{cases}$$

So T is the assignment-wise evaluated conjunction.
The corresponding operators $T_\vee, T_\rightarrow, T_\neg$ and $T_\leftrightarrow$ are defined analogously to $T_\wedge$: assignment-wise evaluated connectives.

This completes the interpretation of Ty2. For the interpretation of Ty2< an additional clause is required.

6. *Ordering*

$$V_{as}(\alpha<\beta) = \begin{cases} 1 & \text{if the world component of } V_{as}(\alpha) \text{ equals the world component of } V_{as}(\beta)\text{, and the time component of } V_{as}(\alpha) \text{ is before the time component of } V_{as}(\beta) \text{ in the linear ordering of T.} \\ 0 & \text{otherwise.} \end{cases}$$

Analogously for $V_{as}(\alpha>\beta)$.

This definition means that in case the world components of α and β are different, then $V_{as}(\alpha<\beta) = 0$. The relation-symbol < does not correspond with a total ordering, and consequently $\neg(\alpha<\beta)$ is not equivalent with $[\alpha>\beta \vee \alpha=\beta]$.

4. PROPERTIES OF Ty2

In the definition of the language Ty2, we introduced a lot of operators (corresponding with connectives and quantifiers). Abstraction was just one among them. In a certain sense, however, it is the most important and powerful operator. The other operators are unnecessary since they can be defined in terms of λ-operators (and =). Also expressions denoting truth values can be defined in this way. I will present the definitions (originating from HENKIN 1963), without further explications because I will not use their details in the sequel. They are presented here for illustrating the central role of λ-abstraction in this system.

4.1. EXAMPLE *definitions based on λ-operators*.
Let $x, y \in VAR_t$ and $f \in VAR_{<t,t>}$. Then

$$T = [\lambda x[x] = \lambda x[x]]$$
$$F = [\lambda x[x] = \lambda x[T]]$$
$$\neg = [\lambda x[F{=}x]]$$
$$\wedge = [\lambda x\lambda y[\lambda f[f(x) = y] = \lambda f[f(T)]]]$$
$$\rightarrow = [\lambda x\lambda y[[x\wedge y] = x]]$$
$$\vee = [\lambda x\lambda y[\neg x\rightarrow y]]$$

Let $\tau \in Ty$ and $z \in VAR_\tau$; then:
$$\forall zA = [\lambda zA = \lambda zT].$$

4.1. END

In certain circumstances, we may simplify formulas of the form $\lambda v[\phi](\alpha)$ by substituting the argument α for the free occurrences of v in ϕ. This kind of simplification is called λ-reduction or λ-conversion. In the theory of λ-calculi this reduction is known under the name β-reduction (α-reduction is change of the bound variable v). I described above the central position of λ-operators. This implies that the prooftheory of Ty2 is essentially the prooftheory of λ-calculus. Therefore it is of theoretical importance to know under what circumstances λ-conversion is allowed. But there is also an important practical motivation. In the next chapters we will encounter frequently formulas with many λ-operators. Then, by λ-conversion, these formulas can be reduced to a manageable size. This practical aspect of reducing formulas is the main motivation for considering λ-conversion here in detail. I start with recalling a theorem which says which kinds of reductions are allowed in all contexts. Thereafter some theorems will be given concerning the reduction of formulas containing λ-operators.

4.2. THEOREM. *Let* $\alpha, \alpha' \in ME_\sigma$ *and* $\beta, \beta' \in ME_\tau$ *such that*
a) β *is part of* α
b) β' *is part of* α'
c) α' *is obtained from* α *by substitution of* β' *for* β.

Suppose that for all as $\in$ AS *holds* $V_{as}(\beta) = V_{as}(\beta')$.
Then for all as $\in$ AS *holds* $V_{as}(\alpha) = V_{as}(\alpha')$.

PROOF. Recall that β is a part of α if in the production process of α some rule is used which has β as one of its arguments. Hence β is used in a construction step $R(..,\beta,..)$, where R is an operator from the algebraic grammar for Ty2. That $V_{as}(\beta) = V_{as}(\beta')$ for all as $\in$ AS, means that β' has the same meaning as β. This means that the present theorem is a reformulation of theorem 6.4 in chapter two (here adapted for the present algebra and the present notion of meaning). Hence the same proof applies.
4.2. END

As a consequence of this theorem, two expressions with the same meaning may be replaced by each other 'salva veritate'. This is a generalization of Leibniz' principle (just as the corresponding theorem in chapter 2) since it applies to formulas of any type to be replaced within formulas of any (other) type. The theorem provides a foundation for all reductions (simplifications) of IL-formulas we will meet in the sequel. A (sub)-formula may be replaced by a formula with the same meaning. This may even be done in case it is not yet known in which larger expression they will occur as subformula. The theorem holds due to the fact that we have an algebraic interpretation of Ty_2.

4.3. DEFINITION. Let $\phi \in ME_\sigma$, $\alpha \in ME_\tau$ and $v \in VAR_\tau$. Then $[\alpha/v]\phi$ denotes the formula obtained from ϕ by *substitution* of α for all free occurrences of v in ϕ. This substitution is defined recursively as follows

$$[\alpha/v]v \underset{d}{=} \alpha,\ [\alpha/v]\,w \underset{d}{=} w \ (\text{if } w \not\equiv v),\ [\alpha/v]c \underset{d}{=} c$$

$$[\alpha/v][\psi = \eta] \underset{d}{=} [[\alpha/v]\psi] = [[\alpha/v]\eta]$$

$$[\alpha/v][\psi(\eta)] \underset{d}{=} [[\alpha/v]\psi]([\alpha/v]\eta)$$

$$\begin{cases}[\alpha/v][\exists w\phi] \underset{d}{=} \exists w[[\alpha/v]\phi] & \text{if } w \not\equiv v\\ [\alpha/v]\exists v\phi \underset{d}{=} \exists v\phi & \end{cases}$$

analogously for $\forall w\phi$ and for $\lambda w\phi$

$$[\alpha/v][\phi\wedge\psi] \underset{d}{=} [[\alpha/v]\phi] \wedge [[\alpha/v]\psi]$$

analogously for the other connectives.

4.3. END

4.4. THEOREM. *Suppose no free variable in* α *becomes bound by substitution of* α *for* v *in* ϕ. *Then* λ*-conversion is allowed; i.e. for all* as $\in$ AS:

$$V_{as}(\lambda v[\phi](\alpha)) = V_{as}([\alpha/v]\phi).$$

PROOF. The clause concerning function application in the definition of Ty2 says:

$$V_{as}(\lambda v[\phi](\alpha)) = V_{as}(\lambda v[\phi])(V_{as}(\alpha)).$$

By definition $V_{as}(\lambda v[\phi])$ is that function which for argument d yields value $V_{[v\rightarrow d]as}(\phi)$. So, writing A for $V_{as}(\alpha)$, we have

$$V_{as}(\lambda v[\phi])(V_{as}(\alpha)) = V_{as}(\lambda v[\phi])(A) = V_{[v\rightarrow A]as}(\phi).$$

We first will prove, that for all as $\in$ AS

$$V_{[v\rightarrow A]as}(\phi) = V_{as}([\alpha/v]\phi).$$

From this equality the proof of theorem easily follows. The proof of the equality proceeds with induction to the construction of ϕ.

1. $\phi \equiv c$, where $c \in \text{CON}_\tau$

$$V_{[v\rightarrow A]as}(c) = F(c) = V_{as}(c) = V_{as}([\alpha/v]c).$$

2. $\phi \equiv w$, where $w \in \text{VAR}_\tau$.

2.1. $w \not\equiv v$

$$V_{[v\rightarrow A]as}(w) = V_{as}(w) = V_{as}([\alpha/v]w).$$

2.2. $w \equiv v$

$$V_{[v\rightarrow A]as}(v) = A = V_{as}(\alpha) = V_{as}[\alpha/v]v.$$

3. $\phi \equiv [\psi = \eta]$

$$V_{[v\rightarrow A]as}(\psi=\eta) = 1 \quad \text{iff} \quad V_{[v\rightarrow A]as}(\psi) = V_{[v\rightarrow A]as}(\eta).$$

By induction hypothesis, this is true iff

$$V_{as}([\alpha/v]\psi) = V_{as}([\alpha/v]\eta),$$

hence iff $V_{as}([\alpha/v][\psi=\eta]) = 1$.

4. $\phi \equiv \lambda w\psi$

4.1. $w \equiv v$

$$V_{[v\to A]as}(\lambda v\psi) = V_{as}(\lambda v\psi) = V_{as}[\alpha/v][\lambda v\psi].$$

4.2. $w \not\equiv v$.

The conditions of the theorem guarantee that w does not occur in α. This fact is used in equality I below. Equality II holds since we may apply the induction hypothesis for assignment $[w\to d]$ as, and equality III follows from the definition of substitution.

$$V_{[v\to A]as}(\lambda w\psi) = V_{[v\to V_{as}(\alpha)]as}(\lambda w\psi) =$$

$$\underline{\lambda} d V_{[w\to d]([v\to V_{as}(\alpha)]as)}(\psi) =_{I} \lambda d V_{[w\to d]([v\to V_{[w\to d]as}(\alpha)]as)}(\psi) =$$

$$\underline{\lambda} d V_{[v\to V_{[w\to d]as}(\alpha)]([w\to d]as)}(\psi) =_{II}$$

$$\underline{\lambda} d V_{[w\to d]as}([\alpha/v]\psi) =$$

$$= V_{as}\lambda w[\alpha/v]\psi =_{III}$$

$$V_{as}[\alpha/v]\lambda w\psi.$$

The proof for $\forall v\psi$ and $\exists v\psi$ proceeds analogously.

5. $\phi \equiv \neg\psi$

$$V_{[v\to A]as}(\neg\psi) = 1 \text{ iff } V_{[v\to A]as}(\psi) = 0$$

by induction hypothesis we have

$$V_{[v\to A]as}(\psi) = V_{as}[\alpha/v]\psi.$$

So $V_{[v\to A]as}(\neg\psi) = 1$ iff $V_{as}\neg[\alpha/v]\psi = 1$.

Analogously for the other connectives.

4.4. END

From theorem 4.2 it follows that in case λ-conversion is allowed on a certain formula, it is allowed in whatever context the formula occurs. So, given a compound formula with several λ-operators, one may first reduce the operators with the smallest scope and so further, but one may reduce also first the operator with the widest scope, or one may proceed in any other sequence. Does this have consequences for the final result? In other words, is there a unique λ-reduced form ('a λ-normal form')? The answer is affirmative. The only reason which prevents a correct application of the λ-conversion is the syntactic constraint that a free variable in α should not become bound by substitution in ϕ. Using α-conversion (renaming of bound variables), this obstruction can be eliminated. It can then be shown that each formula in Ty_2 can be reduced by use of α- and λ-conversion to a λ-reduced form which is unique, up to the naming of bound variables (see the proof for typed λ-calculus in ANDREWS 1971 of PIETRZYKOWSKI 1973, which proof can be applied to Ty_2 as well). This property of reduction system is known under the name 'Church-Rosser property'.

The theorem we proved concerning λ-conversion gives a syntactic description of situations in which λ-conversion is allowed. It is, however, possible that the condition mentioned in the theorem is not satisfied, but that nevertheless λ-conversion leads to an equivalent formula. A semantic description of situations in which λ-conversion is allowed, is given in the theorem below. This semantic description is not useful for simplifying Ty2 formulas, since there are no syntactic properties which correspond with the semantic description in the theorem. In applications for the semantics of natural languages or programming languages we will have additional information (for instance from meaning postulates) which makes it possible to apply this theorem on the basis of syntactic criteria.

4.5. <u>THEOREM</u> (JANSSEN 1980). *Let* $\lambda v[\phi](\alpha) \in ME$, *and suppose that for all* as,as' $\in$ As: $V_{as}(\alpha) = V_{as'}(\alpha)$.
Then for all as $\in$ AS: $V_{as}(\lambda v[\phi](\alpha)) = V_{as}([\alpha/v]\phi)$.

<u>PROOF</u>. Consider the proof of theorem 4.4. The only case where is made use of the fact that no variable in α becomes bound, is in the equality I in case 4. Since the condition for the present theorem requires that the interpretation of α does not depend on the choice of as, we have

$$V_{as}(\alpha) = V_{[w\mapsto d]as}(\alpha).$$

Consequently the proof of 4.4 applies, using this justification for equality I.

4.5. END

Ty2 contains typed λ-calculus as a sub-theory. Since typed λ-calculus is not axiomatizable, Ty2 is not axiomatizable either. That typed λ-calculus is not axiomatizable becomes evident if one realizes that its models are very rich: they contain models for the natural numbers. The formal proof of the non-axiomatizability is based upon standard techniques and seems well known. GALLIN (1975) does not give a reference when remarking that typed λ-calculus is not axiomatizable, and HENKIN (1950) only gives some hints concerning a proof. A sketch of a possible proof is as follows. An effective translation of Peano arithmetic into typed λ-calculus is defined (see below for an example). Then it is proven that every formula ϕ from Peano arithmetic is true in the standard model of natural numbers iff the translation of ϕ is true in all standard models for Ty2. Since arithmetic truth is not axiomatizable, Ty2 cannot be axiomatizable either.

An example of an effective translation of Peano arithmetic into typed λ-calculus is given in CHURCH (1940). For curiosity I mention the translations of some numbers and of the successor operator S. Also the Peano-axioms can be formulated in typed λ-calculus. The formulas translating 0,1,2 and S, contain the variables $x \in \mathrm{VAR}_e$, $f \in \mathrm{VAR}_{<e,e>}$, and $v \in \mathrm{VAR}_{<<<e,e>,e>,e>}$. One easily checks that S(0) = 1 and S(1) = 2.

arithmetics	*translation*
0	$\lambda f \lambda x[x]$
1	$\lambda f \lambda x[f(x)]$
2	$\lambda f \lambda x[f(f(x))]$
S	$\lambda v \lambda f \lambda x[f(v(f)(x))]$.

As I already said in section 1, Ty2 is generalized complete: i.e. the class of formulas valid in all generalized models is axiomatizable. The proof is a simple generalization of the proof for one-sorted type theory (HENKIN 1950). I will define below the generalized models and present the axioms without further proof.

The generalized domains of type $\tau \in \mathrm{Ty}$, denoted GD_τ, are defined by

1. $GD_e = A, \quad GD_s = I, \quad GD_t = \{0,1\}$
2. $GD_{<\sigma,\tau>} \subset GD_\tau^{GD_\sigma} \qquad (\sigma,\tau \in Ty)$.

The generalized meanings of type τ denoted GM_τ, are defined by

$GM_\tau = GD_\tau^{AS}$ where AS is the set of variable assignments.

A generalized model is a pair <GM,F>, where

1. $GM = \cup_\tau GD_\tau$
2. $F \in \mathcal{F}$ where $\mathcal{F}$ is the collection of interpretations of constants
3. the pair <GM,F> is such that there exists a function V which assigns to each formula a meaning, and which satisfies the clauses a,b,1,...6 from the definition of V for Ty2 in section 3.

Without the last requirement concerning generalized models, there might arise difficulties to define V for some model: the interpretation of λ might fail because the required function may fail to belong to the model. The addition of requirement 3 makes that such a situation cannot arise. On the other hand, the third condition makes that it is not evident that generalized models exist since the given definition is not an inductive definition. It can be shown, however, that out of any consistent set of formulas such a model can be built.

The axioms for Ty2 are as follows (GALLIN 1975, p.60):

A1) $g(T) \wedge g(F) = \forall x[g(x)] \qquad x \in VAR_t,\ g \in VAR_{<t,t>}$

A2) $x = y \rightarrow f(x) = f(y) \qquad x \in VAR_\tau,\ f \in VAR_{<\tau,t>}$

A3) $\forall x[f(x) = g(x)] = [f = g] \quad x \in VAR_\sigma,\ f,g \in VAR_{<\sigma,\tau>}$

AS4) $\lambda x[A(x)](B) = [B/x](A(x))$ where the condition of th.4.4 is satisfied.

Furthermore, there is the following rule of inference:

From $A = A'$ and the formula B one may infer formula B', where B' comes from B by replacing one occurrence of A which is part of B, by the formula A' (cf. theorem 4.2).

5. INTENSIONAL LOGIC

The language of intensional logic is for the most part the same as the language Ty2. The collection of types of IL is a subset of the collection types of Ty2. The set T of types of IL (sorts of IL) is defined as the smallest set such that

1. $e \in T$ and $t \in T$
2. if $\sigma,\tau \in T$ then $<\sigma,\tau> \in T$
3. if $\tau \in T$ then $<s,\tau> \in T$.

The language IL is defined by the following algebra.

$$IL = <<[CON_\tau \cup VAR_\tau]_{\tau\in T}, \underline{R} \cup \{R_\vee, R_\wedge, R_\Box, R_W, R_H\}>, t>$$

where

a. CON_τ and VAR_τ are the same as for Ty2 (as far as the types involved belong to T)
b. $\underline{R}$ consists of all the operations of Ty2 (as far as the types involved belong to T).

The new operators are as follows

1. $R_{\vee,\tau}: ME_{<s,\tau>} \to ME_\tau$ defined by $R_{\vee,\tau}(\alpha) = [{}^\vee\alpha]$
2. $R_{\wedge,\tau}: ME_\tau \to ME_{<s,\tau>}$ defined by $R_{\wedge,\tau}(\alpha) = [{}^\wedge\alpha]$
3. $R_\Box: ME_t \to ME_t$ defined by $R_\Box(\phi) = [\Box\phi]$
4. $R_W: ME_t \to ME_t$ defined by $R_W(\phi) = [W\phi]$
5. $R_H: ME_t \to ME_t$ defined by $R_H(\phi) = [H\phi]$.

The symbol $^\vee$ is pronounced as 'extension' or 'down', $^\wedge$ as 'intension' or 'up', $\Box$ as 'necessarily', and W and H are the future tense operator (W ~ 'will'), and the past tense operator respectively (H ~ 'has'). This use of the symbols H and W follows Montague's PTQ. It should be observed that his notation conflicts with the tradition in tense logic, where P('past') and F('future') are used for this purpose. The operators W and H are used in tense logic for respectively 'it always will be the case' and 'it has always be the case'.

The semantics of IL will be defined indirectly: by means of a translation of IL into Ty2<. I will employ the techniques developed in chapter 2 and design a Montague grammar for IL. This means that a homomorphism Tr will be defined from the term algebra T_{IL} associated with IL, to an algebra Der(Ty2) which is derived from Ty2< (see figure 1). The meaning of an IL expression ϕ is then defined as its image under the composition of this translation Tr and the interpretation homomorphism V for Ty_2 (where V is restricted to the derived algebra). This composition Tr∘V will be denoted

V as well, since from the context it will be clear whether the interpretation of an IL expression or of a Ty2 expression is meant.

$$\begin{tikzcd}
 & T_{IL} \arrow[d, "Tr"] \\
Ty2< \arrow[r, dashed, "Der"] \arrow[d, "V"] & Der(Ty2<) \arrow[d, "V"] \\
M \arrow[r, dashed, "Der"] & Der(M)
\end{tikzcd}$$

Figure 1. IL as a Montague grammar

The translation Tr will introduce the variables $v_{1,s}$ and $v_{2,s}$, which will play a special role in the interpretation of IL. Following GROENENDIJK & STOKHOF 1981, these variables will be written a and a' respectively. The translation Tr introduces only variable a as free variable of type s. Since the expressions of IL contain no variables of type s, this means that the interpretation of an IL-expression is determined by the interpretation of the IL-variables and the interpretation of a. Hence the meaning of an IL-expression can be considered as a function with as domain the assignments to pairs consisting of the value of a, and an assignment to IL-variables.

Let the collection G of assignments to IL variables be defined by

$$G = \prod_{\tau} D_{\tau}^{VAR_{\tau}}.$$

The meaning of an IL-formula $\phi \in ME_{\tau}$ is then an element of $D^{I\times G}$, where the component I determines the value assigned to a. The interpretation of α with respect to $i \in I$ and $g \in G$ is denoted by $V_{i,g}(\alpha)$.

From chapter 2, I recall a special method for the description of derived algebras. Let the original algebra be $A = \langle (A_s)_{s\in S}, (F_{\gamma})_{\gamma\in\Gamma}\rangle$. Then a derived algebra is uniquely determined by the following information (see chapter 2, theorem 8.13).

1) A collection S' of sorts of the derived algebra and a mapping $\tau: S' \to S$ which associates new sorts with old sorts.
2) A collection $(H_{s'})_{s'\in S'}$ of generators for the new algebra, such that $H_{s'} \subset A_{\tau(s')}$.
3) A collection $P \subset POL^{A}$ of operators of the new algebra, and a typegiving

function for these operators which has to satisfy certain requirements.

The interpretation of IL will be defined by means of an algebra which is derived from Ty2 in the just mentioned way. The specification of the three components is as follows.

1) The set T of types of IL is a subset of set of types of Ty2<. Hence the set of types of the derived algebra is T. For the mapping τ we take the identity mapping.

2) There are two kinds of generators in the derived algebra: those which correspond with variables of IL, and those which correspond with the constants of IL. As generators corresponding with the IL-variables, the same Ty2-variables are taken. For the constants we do not proceed in this way. Would we have done so, then a constant of IL would always been associated with one and the same semantical object, because the Ty2-constants have this property. For applications this is not desirable. For instance, the constant *walk* will be interpreted (at a given index) as a function from entities to truth values, thus determining the set of entities walking on that index. We desire, however, that for another index this set may be different. Therefore it is not attractive to take constants of Ty2< as generators of the derived algebra. We will use the variable $a \in VAR_s$ for indicating the current index. The generator corresponding to the IL-constant $c_{n,\sigma}$ is the compound formula $c_{n,<s,\sigma>}(a)$. Note that this formula contains the constant with the same index, but of one intension level higher. These considerations explain the following definition

$$H_{\sigma'} = VAR_{\sigma'} \cup \{c_{n,<s,\sigma'>}(a) \mid n \in \mathbb{N}\}.$$

3) Note first that the type giving function for the polynomials does not need to be specified because all types of IL are types of Ty2. There are two kinds of operators. Some operators of IL are also operators of Ty2< as well. The polynomial symbols for these operators (see chapter 2, remark after theorem 4.6) are incorporated in P. The polynomial symbols corresponding with the other operators of IL are as follows

$R_{\vee,\tau}$: $x_{1,\tau}(a)$

$R_{\wedge,\tau}$: $\lambda a[x_{1,\tau}]$

$R_{\Box}$: $\forall a[x_{1,t}]$

R_W: $\exists a'>a[\lambda a[X_{1,t}](a')]$

R_H: $\exists a'<a[\lambda a[X_{1,t}](a')]$.

In the polynomials for $R_{\Box}$ and $R_{\wedge}$ a binder for *a* is introduced. In most cases this does not give rise to vacuous quantification or abstraction since the variable *X* will often be replaced by an expression containing a free variable *a* introduced by the translation of some constant. The polynomial for R_W might be read as $\exists a'>a[[a'/a]X_{1,t}]$ and for R_H analogously (but these expressions are not polynomial symbols).

The information given above completely determines a unique derived algebra. Theorems of chapter 2 guarantee that in this indirect way the interpretation of IL is defined as the composition of the translation of IL into Der(Ty2<) with the interpretation of this derived algebra.

6. PROPERTIES OF IL

Below, and in the next section, some theorems concerning IL will be presented. The proofs will rely on the way in which the meaning of IL is defined: as the composition of the translation homomorphism Tr and the meaning homomorphism V for Ty2<. Hence the interpretation $V_{i,g}(\phi)$ of an IL-expression ϕ equals the interpretation $V_{as}(Tr(\phi))$ of its translation in Ty2<, where as is an assignment to Ty2-variables such that as(*a*) = i and as(*v*) = g(*v*) for all IL-variables *v*. Hence we may prove $V_{i,g}(\phi) = V_{i,g}(\psi)$ by proving $V_{as}(Tr(\phi)) = V_{as}(Tr(\psi))$ for such a Ty-assignment as. If η is an expression of Ty2, then the notation $V_{i,g}(\eta)$ will be used for $V_{as}(\eta)$, where as is an arbitrary assignment to Ty2 variables such that as(a) = i and as(v) = g(v) for all IL-variables v. If ϕ is of type t, we will often write $i,g \models \phi$ instead of $V_{i,g}(\phi) = 1$. When i or g are arbitrary, they will be omitted. Hence $\models \phi$ means that for all i and g it is the case that $i,g \models \phi$.

In section 4 we notified the theoretical importance of λ-conversion for Ty2: all quantifiers and connectives can be defined by means of lambda operators. For Ty2 the same holds, so it is of theoretical importance to know under which circumstances λ-conversion is allowed. But there also is an important practical motivation. We will frequently use λ-conversion for simplifying formulas. For these reasons I will consider the IL-variants of the theorems concerning the substitution of equivalents and concerning λ-conversion.

6.1. <u>THEOREM</u>. *Let* $\alpha,\alpha' \in ME_\sigma$ *and* $\beta,\beta' \in ME_\tau$ *such that*

a) β *is part of* α

b) β' *is part of* α'

c) α' *is obtained from* α *by substitution of* β' *for* β.

Suppose that for all $i \in I$ *and* $g \in G$

$$V_{i,g}(\beta) = V_{i,g}(\beta').$$

Then for all $i \in I$ *and* $g \in G$

$$V_{i,g}(\alpha) = V_{i,g}(\alpha').$$

<u>PROOF</u>. This theorem could be proven in the same way as the corresponding theorem for Ty2: by reference to theorem 6.4 from chapter 2. I prefer, however, to prove the theorem by means of translation into Ty2 because this shows some arguments which will be used (implicitly or explicitly) in the other proofs.

From (1) we may conclude that (2) holds, from which (3) immediately follows:

(1) for all $i \in I$, $g \in G$: $V_{i,g}(\beta) = V_{i,g}(\beta')$

(2) for all $i \in I$, $g \in G$: $V_{i,g}(Tr(\beta)) = V_{i,g}(Tr(\beta'))$

(3) for all $as \in As$: $V_{as}(Tr(\beta)) = V_{as}(Tr(\beta'))$.

Recall that β is a part of α if there is in the production of α an application $R(..,\beta,..)$ of an operator R with β as one of its argument. Since Tr is a homomorphism defined on such production processes, it follows that $Tr(\alpha) = Tr(..,R(..,\beta,..)..) = ...R'(..,Tr(\beta),..)...$ (here is R' the polynomial operator over Ty2 which corresponds with the IL-operator R). This says that the translation of a part of α is a part of the translation of α. Consequently we may apply to (3) theorem 4.2 and conclude that (4) holds.

(4) For all $as \in AS$: $V_{as}(Tr(\alpha)) = V_{as}(Tr(\alpha'))$.

From this follows

(5) For all $i \in I$, $g \in G$: $V_{i,g}(\alpha) = V_{i,g}(\alpha')$.

6.1. END

An important class of expressions are the expressions which contain neither constants, nor the operators $^\vee$, H or W. Such expressions are called modally closed; a formal definition is as follows.

6.2. <u>DEFINITION</u>. An expression of IL is called *modally closed* if it is an element of the subalgebra

$$<[(VAR_\tau)_{\tau\in Ty}], \underline{R} \cup \{R_\wedge, R_\Box\}>$$

where $\underline{R}$ consists of the operators of Ty2.
6.2. END

The theorem for λ-conversion which corresponds with theorem 4.4 reads as follows

6.3. <u>THEOREM</u>. *Let* $\lambda v[\phi](\alpha) \in ME_\tau$, *and suppose that no free variable in* α *becomes bound by substitution of* α *for* v *in* ϕ. *Suppose that one of the following two conditions holds:*
1. *no occurrence of* v *in* ϕ *lies within the scope of* $^\wedge$,H,W, *or* $\Box$
2. α *is modally closed.*
Then for all $i \in I$ *and* $g \in G$

$$i,g \models \lambda v[\phi](\alpha) = [\alpha/v]\phi.$$

<u>PROOF</u>. *Part 1.*
Suppose condition 1 is satisfied.
The translation $Tr(\alpha)$ of α contains the same variables as α, except for the possible introduction of variables of type s. The translation $Tr(\phi)$ of ϕ contains the same binders as ϕ since only $^\wedge$,H,W, and $\Box$ introduce new binders (see the definition of Tr). Since ϕ itself does not contain binders for variables of type s, we conclude that:
No free variable in $Tr(\phi)$ becomes bound by substitution of $Tr(\alpha)$ for v in $Tr(\phi)$.
Theorem 6.1 allows us to conclude from this that for all $as \in AS$.

$$V_{as}(\lambda v[Tr(\phi)](Tr(\alpha))) = V_{as}([Tr(\alpha)/v]Tr(\phi)).$$

Note that $Tr(\phi)$ has the same occurrences of v as ϕ, hence one easily proves with induction that

$$[Tr(\alpha)/v]Tr(\phi) = Tr([\alpha/v]\phi).$$

Consequently $V_{as}(Tr(\lambda v[\phi](\alpha))) = V_{as}(Tr([\alpha/v]\phi))$.

So $\mathrm{Tr}\circ V(\lambda v[\phi](\alpha)) = \mathrm{Tr}\circ V([\alpha/v]\phi)$.
From this it follows that for all $g \in G$ and $i \in I$

$$g,i \models \lambda v[\phi](\alpha) = [\alpha/v]\phi.$$

Part 2.
Suppose that condition 2 is satisfied.
The translation of ϕ may introduce binders for variables of type s, but it does not introduce binders for variables of other types (see the definition of Tr). The expression α does not contain free variables of type s, and the translation in this case does not introduce such variables since the only kind of expressions which give rise to new free variables are constants, and the operators $^{\vee}$,H, and W. So we may conclude that:

No free variable in $\mathrm{Tr}(\alpha)$ becomes bound by substitution of $\mathrm{Tr}(\alpha)$ for v in $\mathrm{Tr}(\phi)$.

From this we can prove the theorem in the same way as we did for the first condition.
6.3. END

In theorem 4.5 a semantic description was given of situations in which λ-conversion is allowed. The IL variant of this theorem reads as follows.

6.4. THEOREM (IL). *Let* $\lambda v[\phi](\alpha) \in \mathrm{ME}$ *and suppose for all* $i,j \in I$ *and* $g,h \in G$: $V_{i,g}(\alpha) = V_{j,h}(\alpha)$. *Then for all* $i \in I$ *and* $g \in G$:

$$V_{i,g}(\lambda v[\phi](\alpha)) = V_{i,g}([\alpha/v]\phi).$$

PROOF. By translation into Ty2 and application of theorem 4.5.
6.4. END

In the light of the role of λ-conversion, it is interesting to know whether λ-conversion in IL has the Church-Rosser property, i.e. whether there is an unique lambda-reduced normal form for IL. In much practical experience with intensional logic I learned that it does not matter in which order a formula containing several λ-operators is simplified: first applying λ-reduction to the most embedded operators, or first the most outside ones, the final result was the same. It was a big surprise that FRIEDMAN & WARREN (1980b) found an IL-expression where different reduction sequences yield different final results. Their example is

$$\lambda x[\lambda y[{}^{\wedge}y = u(x)](x)](c)$$

where x and y are variables of some type τ, c a constant of type τ, and u a variable of type $<\tau,<s,\tau>>$. For each of the λ operators the conditions for the theorem are satisfied. Reducing first λx yields

$$\lambda y[{}^{\wedge}y = u(c)](c)$$

which cannot be reduced further since the conditions for λ-conversion are not satisfied. Reducing first λy yields

$$\lambda x[{}^{\wedge}x = u(x)](c)$$

which cannot be reduced either. We end up with two different, although logical equivalent, formulas; i.e. there is no λ-reduced normal form for IL.

The example depends on the particular form for λ-contraction: for all occurrences of the variable the substitution takes place in one and the same step. FRIEDMAN & WARREN (1980
is equivalent to

$$[\lambda x[{}^{\wedge}x]](c) = u(c).$$

This formula is in some sense further reduced. They conjecture that for a certain reformulation of λ-conversion the Church-Rosser property could be provable.

It is interesting to compare the above discussion with the situation in Ty2, where there is a unique λ-reduced form. The Ty2-translation of the Friedman-Warren formula is ($c' \in CON_{<s,\tau>}$!)

$$\lambda x[\lambda y[\lambda a[y] = u(x)](x)]c'(a).$$

This reduces to

$$\lambda y[\lambda a[y] = u(c'(a))]c'(a).$$

After renaming the bound variable a to i, this reduces further to

$$\lambda y[\lambda i[c'(a)] = u(c'(a))].$$

Note that this last reduction is possible here (and not in IL) because of the explicit abstraction λi, instead of the implicit abstraction in $^{\wedge}y$.

A lot of laws of IL are variants of well known laws for predicate logic and type logic. An exception to this description is formed by alws involving constants. The constants of IL are interpreted in a remarkable way: their interpretation is state dependent. Invalid is, for instance, the existential generalization $\Box A(a) \rightarrow \exists x \Box A(x)$, whereas $\forall y(\Box A(y) \rightarrow \exists x \Box A(x))$ is valid. Invalid is $\forall x[x = c \rightarrow \Box[x = c]]$, whereas $\forall x \forall y[x = y \rightarrow \Box[x = y]]$ is valid. Other examples of invalid formulas are $\exists y \Box[y = c]$ and $\forall x[\Diamond[A(x)] \rightarrow \Diamond A(a)]$, where $\Diamond$ abbreviates $\neg \Box \neg$.

Since IL contains type theory as a sublanguage, there is no axiomatization of IL (see also section 4). But IL is generalized complete as is proved by GALLIN (1975). The proof is obtained by combining the proof of generalized completeness of type theory (HENKIN 1950), and the completeness proof for modal logic (see HUGHES & CRESSWELL 1968). The following axioms for IL are due to GALLIN (1975); the formulation is adapted

A1	$[g(T) \wedge g(F)] = \forall x[g(x)]$	$x \in VAR_t$, $g \in VAR_{<t,t>}$
A2	$x = y \rightarrow f(x) = f(y)$	$x \in VAR_\sigma$, $f \in VAR_{<\sigma,t>}$
A3	$\forall x[f(x) = g(x)] = [f = g]$	$x \in VAR_\sigma$, $f,g \in VAR_{<\sigma,\tau>}$
A4	$\lambda x[\alpha](\beta) = [\beta/x]\alpha$	if the conditions of theorem 5.2. are satisfied
A5	$\Box[^{\vee}f = {}^{\vee}g] = [f = g]$	$f,g \in VAR_{<s,\tau>}$
A6	$^{\vee\wedge}\alpha = \alpha$	$\alpha \in ME_\sigma$.

The rule of inference is:

From $A = A'$ and the formula B one may infer to formula B', where B' comes from B by replacing one occurrence of A, that is part of B, by A'

Notice that the translation of axiom A5 into Ty2, would lead to a formula of the form of A3, and that the translation of A6 into Ty2 would be of the form of A4. Axiom A6 will be considered in detail in the next section.

I will not consider details of this axiomatization for the following three reasons.

1) This axiomatization was designed for constituting a basis for the completeness proof, and not for proving theorems in practice. To prove the most simple theorems on the basis of the above axioms would be rather difficult. All proofs that will be given in the sequel are semantic proofs and not syntactic proofs: i.e. the proofs will be based upon the interpretation of formulas and not on axioms.
2) We will work with models which obey certain postulates. These postulates express many important semantic details, and most of the proofs we are interested in, are based upon these special properties and not on the general properties described by the axioms.
3) We do not work with generalized models,but with standard models. So the axiomatization is not complete in this respect.

7. EXTENSION AND INTENSION

In this section special attention is paid to the interaction of the extension operator and the intension operator. In this way some insight is obtained in these operators and their sometimes remarkable properties. The 'Bigboss' example, which will be given below, is important since the *Bigboss* will figure as (counter)example on several occasions.

7.1. THEOREM. *For all* i,g: $V_{i,g}{}^{\vee\wedge}[\alpha] = V_{i,g}\alpha$.

PROOF. $\mathrm{Tr}({}^{\vee\wedge}\alpha) = \mathrm{Tr}({}^{\wedge}\alpha)(a) = \lambda a[\mathrm{Tr}(\alpha)](a) = [a/a]\mathrm{Tr}(\alpha) = \mathrm{Tr}(\alpha)$.

Note that λ-conversion is allowed because the condition of theorem 4.5 is satisfied.

7.1. END

It was widely believed that the extension operator should be the right inverse of the intension operator as well. This believe is expressed in PARTEE (1975,p.250) and in GEBAUER (1978,p.47). It is true, however, only in certain cases. In order to clarify the situation, consider the following description of the effect of ${}^{\wedge\vee}$. Let I and D be denumerable, so $D_\tau = \{d_1,d_2,\dots\}$ and $I = \{i_1,i_2,\dots\}$. Let $\alpha \in ME_\tau$, hence the meaning of α is a function with domain I and range D_τ. We may represent α as a denumerable sequence of elements from D_τ. An example is given in figure 2.

arguments	i_1	i_2	i_3	i_4	..
$V_{i_1,g}(\alpha)$: values for the respective arguments	$\underline{d_1}$	d_2	d_1	d_3	..
$V_{i_2,g}(\alpha)$: values for the respective arguments	d_2	$\underline{d_2}$	d_3	d_1	..
$V_{i_3,g}(\alpha)$: values for the respective arguments	d_1	d_1	$\underline{d_2}$	d_1	..

Figure 2. The interpreation of $\alpha \in ME_\tau$

The interpretation for index i of $^{\wedge\vee}\alpha$ is some function with domain I and range D_τ. Which function it is does not depend on the choice of i, because $Tr(^{\wedge\vee}\alpha)$ which equals $\lambda a[Tr(\alpha)(a)]$, contains no free variables of type s. The function $V_{i,g}(^{\wedge\vee}\alpha)$ yields for argument i_n as value the value of $V_{i_n,g}(\alpha)$ for argument i_n. So in the above example for argument i_1 it yields as value d_1, for i_2 it yields d_2, and for i_3 it yields d_2 (the underlined elements). One observes that $^{\wedge\vee}\alpha$ is the diagonalization of α. So $^{\wedge\vee}\alpha = \alpha$ will hold for all i,g if for all $i,j \in I$: $V_{i,g}(\alpha) = V_{j,g}(\alpha)$. A syntactic description of a class of formulas for which the equality holds, is given in the following theorem.

7.2. THEOREM. *Suppose* α *is modally closed. Then for all* i,g: $V_{i,g}[^{\wedge\vee}\alpha] =$
$= V_{i,g}(\alpha)$.

PROOF. The functions $Tr(^{\wedge\vee}\alpha)$ and $Tr(\alpha)$ denote functions with domain I, and their values for an arbitrary argument i are the same:

$$Tr(^{\wedge\vee}\alpha)(i) = \lambda a[Tr(\alpha)(a)](i) = [i/a]Tr(\alpha)([i/a]a) = Tr(\alpha)(i).$$

Notice that $[i/a]Tr(\alpha) = Tr(\alpha)$ since α is modally closed. So for all $as \in AS$: $V_{as}\ Tr(^{\wedge\vee}\alpha) = V_{as}(Tr(\alpha))$, which proves the theorem.
7.2. END

The insights obtained from the 'diagonalization' point of view, can be used to obtain a counterexample for the case that α is not modally closed. It suffices to find an expression α of type τ which has at index i_1 as its denotation a constant function from I to D_τ, say with constantly value d_1, and at index i_2 as its denotation a constant function yielding some other value, say d_2. This situation is represented in figure 3.

arguments	i_1	i_2	i_3	i_4
$V_{i_1,g}(\alpha)$: values for the respective arguments	$\underline{d}_1$	d_1	d_1	d_1
$V_{i_2,g}(\alpha)$: values for the respective arguments	d_2	$\underline{d}_2$	d_2	d_2

Figure 3. A counterexample for ${}^{\wedge\vee}\alpha$.

Now $V_{i_1,g}[{}^{\wedge\vee}\alpha](i_1) = d_1 \neq d_2 = V_{i_2,g}[{}^{\wedge\vee}\alpha](i_1)$.

One way to obtain this effect is by means of a constant. I give an example of a somewhat artificial nature (due to JANSSEN 1980). Let the valuation of the constant *Bigboss* $\in CON_{<s,e>}$ for index i be the function constantly yielding the object $d \in D_e$ to which the predicate 'is the most powerful man on earth' applies on that index. A possible variant of this example would be the constant *Miss-world* $\in CON_{<s,e>}$, to which the predicate applies 'is elected as most beautiful woman in the world'.

Assume that for the constant *Bigboss* holds that for all $j \in I$ both

$$V_{i_1,g}[\mathit{Bigboss}](j) = V_{i_1,g}[\mathit{Reagan}]$$

and

$$V_{i_2,g}[\mathit{Bigboss}](j) = V_{i_2,g}[\mathit{Bresnjev}].$$

Then

$$V_{i_1,g}[{}^{\wedge\vee}\mathit{Bigboss}](i_2) = \underline{\lambda} i[V_{i,g}[\mathit{Bigboss}](i)](i_2) =$$

$$= V_{i_2,g}[\mathit{Bigboss}](i_2) = V_{i_2,g}[\mathit{Bresnjev}].$$

So

$$V_{i_1,g}[{}^{\wedge\vee}\mathit{Bigboss}](i_2) \neq V_{i_1,g}[\mathit{Bigboss}](i_2).$$

This effect does not depend on the special interpretations for constants. Another way to obtain the desired effect is by taking for α the expression x where x is a variable of type <s,<s,e>>. Let $g(x)$ be defined such that for all $j \in I$: $g(x)(j) = V_{j,g}[\mathit{Bigboss}]$. Then ${}^{\wedge\vee\vee}x \neq {}^{\vee}x$:
because $V_{i_1,g}({}^{\wedge\vee\vee}x)(i_2) = \underline{\lambda} i[g(x)(i)(i)](i_2) = V_{i_2,g}[\mathit{Bigboss}](i_2) =$
$= V_{i_2,g}[\mathit{Bresnev}]$
whereas $V_{i_1,g}({}^{\vee}x)(i_2) = g(x)(i_1)(i_2) = V_{i_1,g}[\mathit{Bigboss}](i_2) = V_{i_1,g}[\mathit{Reagan}]$.

The next example concerns the situation that IL is extended with the *if-then-else* construct. Let β be of type t and φ and ψ of type τ, and define

$$V_{i_1,g}[\underline{if}\ \beta\ \underline{then}\ \phi\ \underline{else}\ \psi] = \begin{cases} V_{i_1,g}(\phi) & \text{if } V_{i_1,g}(\beta) = 1 \\ V_{i,g}(\psi) & \text{otherwise.} \end{cases}$$

Let x and y be variables of type <s,e> and assume that $g(x) \neq g(y)$, but that for some i holds that $g(x)(i) = g(y)(i)$.
Then it is not true that for all i,g

$$i,g \models {}^{\wedge\vee}[\underline{if}\ {}^{\vee}x = {}^{\vee}y\ \underline{then}\ x\ \underline{else}\ y] = [\underline{if}\ {}^{\vee}x = {}^{\vee}y\ \underline{then}\ x\ \underline{else}\ y].$$

This kind of expression is rather likely to occur in the description of semantics of programming languages.

The last example is due to GROENENDIJK & STOKHOF (1981). They consider the semantics of *whether*-complements. An example is

John knows whether Mary walks.

The verb *know* is analysed as a relation between an individual and a proposition. Which proposition is John asserted to know? If it is the case that Mary walks, then John is asserted to know that Mary walks. And if Mary does not walk, then he is asserted to know that Mary does not walk. So the proposition John knows appears to be

$\underline{if}\ walk(mary)\ \underline{then}\ {}^{\wedge}walk(mary)\ \underline{else}\ {}^{\wedge}[\neg walk(mary)]$.

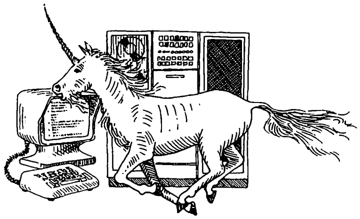
This example provides for a rather natural example of a formula φ for which ${}^{\wedge\vee}\phi$ does not reduce.

CHAPTER IV

MONTAGUE GRAMMAR AND PROGRAMMING LANGUAGES

ABSTRACT

The present chapter starts with an introduction to the semantics of programming languages. The semantics of the assignment statement is considered in detail, and the traditional approaches which use predicate transformers are shown to give rise to problems. A solution is presented according to the algebraic framework defined in the first chapters of this book; it uses an extension of intensional logic.



1. ASSIGNMENT STATEMENTS

1.1. Introduction

Programs are pieces of text, written in some programming language. These languages are designed for the special purpose of instructing computers. They also are used in communication among human beings for telling them how to instruct computers or for communicating algorithms which are not intended for computer execution. So for programming languages we are in the same situation as for natural languages. We have a syntax and we have intended meanings, and we wish to relate these two aspects in a systematic way. Since we are in the same situation, we may apply the same framework. In this chapter we will do so for a certain fragment of the programming language ALGOL 68.

There exists nowadays several thousands of mutually incompatible programming languages. They are formal languages with a complete formal definition of the syntax of the language. Such a definition specifies exactly when a string of symbols over the alphabet of the language is a program and when not. The definition of a programming language also specifies how a program should be executed on a computer, or, formulated more generally, what the program is intended to do. In fact, however, several programming languages are not adequately documented in this respect. Each programming language has its own set of strange idiosyncracies, design errors, perfectly good ideas and clumsy conventions. However, there are a few standard types of instructions present in most of the languages. The present chapter deals mainly with the semantics of one of those instructions: the assignment statement which assigns a value to a name.

It appears that assignment statements exhibit the same phenomena as intensional operators in natural languages. A certain position in the context of an assignment statement is transparent (certain substitutions for names are allowed), whereas another position is opaque (such substitutions are not allowed). The traditional ways of treating the semantics of programming languages do not provide tools for dealing with intensional phenomena. A correct treatment of simple cases of the assignment statement can be given, but for the more complex cases the traditional approaches fail. I will demonstrate that the treatment of intensional operators in natural language, as given in the previous chapters, may also be applied to programming languages, and that in this way a formalized semantics of assignment statements can be given which deals correctly with the more complex cases as well. Hence we will use the same logic: intensional logic (see chapter 3). The idea to use this logic goes back to JANSSEN & Van EMDE BOAS (1977a,b). We will however, not only use the same logic, but also the same compositional, algebraic framework. In chapter 1 the background of this framework was discussed, and in chapter 2 it was defined formally and compared with the algebraic approach of Adj. For a bibliography of universal algebraic and logical approaches in computer science see ANDREKA & NEMETI 1969. The first sections of the present chapter are a revision of JANSSEN & Van EMDE BOAS (1981).

1.2. Simple assignments

One may think of a computer as a large collection of *cells* each containing a *value* (usually a number). For some of these cells names are available in the programming language. Such names are called *identifiers* or, equivalently, *variables*. The term 'identifier' is mainly used in contexts dealing with syntax, 'variable' in contexts dealing with semantics. The connection of a variable with a cell is fixed at the start of the execution of a program and remains further unchanged. So in this respect a variable does not vary. However, the cell associated with a variable stores a value, and this value may be changed several times during the execution of a program. So in this indirect way a variable can vary. The *assignment statement* is an instruction to change the value stored in a cell.

An example of an assignment statement is: $x := 7$, read as 'x becomes 7'. Execution of this assignment has the effect that the value 7 is placed in the cell associated with x. Let us assume that initially the cells associated with x, y and w contain the values 1, 2 and 4 respectively (figure 1a). The execution of $x := 7$ results in the situation shown in figure 1b. Execution of $y := x$ has the effect that the value stored in the cell associated with x is copied in the cell associated with y (figure 1c). The assignment $w := w + 1$ applied in turn to this situation, has the effect that the value associated with w is increased by one (figure 1d).

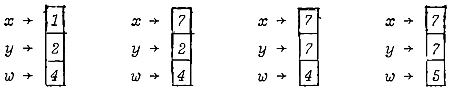


Figure 1a Initial Situation — Figure 1b After $x := 7$ — Figure 1c After $y := 7$ — Figure 1d After $w := w + 1$

Now the necessary preparations are made for demonstrating the relation with natural language phenomena. Suppose that we are in a situation where the identifiers x and y are both associated with value 7. Consider now the assignment

(1) $x := y + 1$.

The effect of (1) is that the value associated with x becomes 8. Now replace identifier y in (1) by x:

(2) $x := x + 1$.

Again, the effect is that the value associated with x becomes 8. So an identifier on the right hand side of ':=' may be replaced by another which is associated with an equal value, without changing the effect of the assignment. One may even replace the identifier by (a notation for) its value:

(3) $x := 7 + 1$.

Replacing an identifier on the left hand side of ':=' has more drastic consequences. Replacing x by y in (1) yields:

(4) $y := y + 1$.

The value of y is increased by one, whereas the value associated with x remains unchanged. Assignment (1), on the other hand, had the effect of increasing the value of x by one; likewise both (2) and (3). So on the left hand side the replacement of one identifier by another having the same value is not allowed. While (2) and (3) are in a certain sense equivalent with (1), assignment (4) certainly is not. Identifiers (variables) behave differently on the two sides of ':='.

It is striking to see the analogy with natural language. I mention an example due to QUINE (1960). Suppose that, perhaps as result of a recent appointment, it holds that

(5) *the dean* = *the chairman of the hospital board.*

Consider now the following sentence:

(6) *The commissioner is looking for the chairman of the hospital board.*

The meaning of (6) would not be essentially changed if we replaced *the commissioner* by another identification of the same person; a thus changed sentence would be true in the same situations as the original sentence. But consider now (7).

(7) *The commissioner is looking for the dean.*

Changing (6) into (7) does make a difference: it is conceivable that the commissioner affirms (6) and simultaneously denies (7) because of the fact that he has not been informed that (5) recently has become a truth. Sentence (7) is true in other situations than sentence (5). Hence they have a different meaning. In the terminology for substitution phenomena, the subject position of *is looking for* is called (*referentially*) *transparent*, and its object position (*referentially*) *opaque* or *intensional position*. Because of the close analogy, we will use the same terminology for programming languages, and call the right hand side of the assignment 'transparent', and its left hand side 'opaque' or 'intensional'.

The observation concerning substitutions in assignments statements, as considered above, is not original. It is, for instance, described in TENNENT 1976 and STOY 1977 (where the term 'transparent' is used) and in PRATT 1976 (who used both 'transparent' and 'opaque'). The semantic treatments of these phenomena which have been proposed, are, however, far from ideal, and in fact not suitable for assignments which are less simple than the ones above. The authors just mentioned, like many others, avoid these difficulties by considering a language without the more complex constructions.

1.3. Other assignments

Above we only considered assignments involving cells which contain an integer as value. In this section I will describe two other situations: cells containing an identifier as value (pointers) and rows of cells (arrays).

Some programming languages also allow for handling cells which contain a variable (identifier) as value (e.g. the languages Pascal and Algol-68). Names of such cells are called *pointer identifiers* or equivalently *pointer variables*, shortly *pointers*. The situation that pointer p has the identifier x as its value, is shown in figure 2a. In this situation, p is indirectly related to the value of x, i.e. 7. The assignment $p := w$ has the effect that the value stored in p's cell becomes w (figure 2b). Thus p is indirectly related to the value of w: the integer 5. When next the assignment $w := 6$ is executed, the integer value indirectly associated with p becomes 6 (figure 2c). So an assignmet can have consequences for pointers which are not mentioned in the assignment statement itself: the value of the variable associated with the pointer may change.

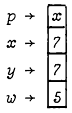


Figure 2a
Initial Situation

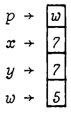


Figure 2b
After $p := w$

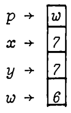


Figure 2c
After $w := 6$

In a real computer, a cell does not contain an integer or a variable, but rather a code for an integer or an code for a variable. For most real computers it is not possible to derive from the contents of a cell, whether it should be interpreted as an integer code or a variable code. In order to prevent the unintended use of an integer code for a variable code, or vice versa, some programming languages (e.g. Pascal) require for each identifier a specification of the kind of values to be stored in the corresponding cells. The syntax of such a programming language then prevents unintended use of an integer code for an identifier code (etc.) by permitting only programs in which each identifier is used for a single kind of value. Other languages leave it to the discretion of the programmer whether to use an identifier for only one kind of value (e.g. Snobol-4). Our examples are from a language of the former type: ALGOL 68.

The programming language ALGOL 68 also allows for higher order pointers, such as pointers to pointers to variables for integer values. They are related to cells which contain as value (the code of) a pointer of the kind described above. These higher order pointers will be treated analogously to the pointers to integer identifiers.

Several programming languages have names for rows of cells (arrays of cells). Names of such rows are called array identifiers, or equivalently array variables. An individual cell can be indicated by attaching a subscript to the array identifier. The element of an array a associated with subscript i is indicated by $a[i]$. The cells of an array contain values of a certain kind: the cells of an integer array contain integers (see figure 3a), and the cells of an array of pointers contain pointers. The execution of the assignment $a[2] := 2$ has the effect that in the cell indicated by $a[2]$ the value 2 is stored (see figure 3b). The subscript may be a complex integer expression. The effect of the assignment $a[a[1]] := 2$ is that the value in $a[1]$ is determined, it is checked whether the value obtained (i.e. 1) is an acceptable index for the array and the assignment $a[1] := 2$

is performed (figure 3c). In the sequel I, will assume that all integers are acceptable indices for subscripts for an array, i.e. that all arrays are of infinite length (of course an unrealistic assumption; but I am interested in formalizing other aspects of arrays). Other kinds of assignment which involve arrays are in the fragment (e.g. the assignment of the whole array in a single action), but I will deal primarlily with assignments of the form just discussed.

$$a \left\{ \begin{array}{l} a[1] \rightarrow \boxed{1} \\ a[2] \rightarrow \boxed{1} \\ a[3] \rightarrow \boxed{8} \\ a[4] \rightarrow \boxed{2} \end{array} \right. \qquad a \left\{ \begin{array}{l} a[1] \rightarrow \boxed{1} \\ a[2] \rightarrow \boxed{2} \\ a[3] \rightarrow \boxed{8} \\ a[4] \rightarrow \boxed{2} \end{array} \right. \qquad a \left\{ \begin{array}{l} a[1] \rightarrow \boxed{2} \\ a[2] \rightarrow \boxed{2} \\ a[3] \rightarrow \boxed{8} \\ a[4] \rightarrow \boxed{2} \end{array} \right.$$

Figure 3a — Initial Situation

Figure 3b — After $a[2] := 2$

Figure 3c — After $a[a[1]] := 2$

2. SEMANTICS OF PROGRAMS

2.1. Why?

Let us consider, as an example, a program which computes solutions of the quadratic equation $ax^2 + bx + c = 0$. The program is based upon the well-known formula

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \quad . \tag{8}$$

The program reads as follows:

```
1.  begin real a, b, c, disc, d, x1, x2;
2.  read ((a,b,c));
3.  disc := b*b - 4*a*c;
4.     d := sqrt (disc);
5.    x1 := -b + d; x1 := x2/(2*a);
6.    x2 := -b - d; x2 := x2/(2*a);
7.  print ((a,b,c,x1,x2,newline))
8.  end.
```

The first line of the program says that the identifiers mentioned there, will only be used as names of locations containing real numbers as values (e.g. *3.14159*). The second and seventh line illustrate that the computer may obtain data from outside (input) and communicate results to the outside world (output). The program also shows that the mathematical formula looks much more compact than the program, but that this compactness is made possible by the use of some conventions which have to be made explicit for the computer. For example, in the program we must write $4*a*c$ for 4 times a times c, while in the formula $4ac$ suffices. In the formula we use two dimensional features, which are eliminated in the program (*sqrt*(..) instead of $\sqrt{...}$). This linear character is necessitated by the fact that programs have to be communicated by way of a sequential channel; for example, the wire connecting the computer with a card reader. The symbol *real* indicates that the identifiers mentioned may only be associated with real values, and the symbols *begin* and *end* indicate the begin and the end of the program.

There exists a considerable confusion among programmers, theoreticians, and designers as to what we should understand by the semantics of a programming language. There are, however, some properties of programs for which there is a measure of agreement on the need for a treatment within the field of semantics. These properties are:

correctness: A program should perform the task it is intended to perform. For example the program given above is incorrect: it does not account for $a = 0$ or $disc < 0$.

equivalence: Two different programs may yield the same results in all circumstances. For example, in the program under discussion we may interchange the order of the computation of $x1$ and $x2$, but we cannot compute d before we compute $disc$.

termination: If we start the execution of a program, will it ever stop? It might be the case that the computer keeps on trying to find the square root of -1, and thus for certain values of a, b and c never halts.

Each of the above properties tells us something about the possible computations the program will perform when provided with input data. We want to predict what may happen in case ...; more specifically, we want to prove that our predictions about the capabilities of the program are correct. How can we achieve this goal? Clearly it is impossible to try out all possible computations of the program, instead one is tempted to run the program on a 'representative' set of input data. This activity is known as program debugging. This way one may discover errors, but one can never *prove* the program to be correct. Still, in practice, most programs used nowadays have been verified only in this way. One might alternatively try to understand the program simply by reading its text. Again this is not of great help, since mistakes made by the programmer can be remade by the reader. The only way out is the invention of a mathematical theory for proving correctness, equivalence, termination etc.. We need a formalized semantics on which such a theory can be based.

2.2. How?

What does a formal semantics for a program look like? The most common approach is a so-called *operational semantics*. One defines the meaning of a program by first describing some abstract machine (a mathematical model of an idealized computer) and next specifying how the program is to be executed on the abstract machine. Needless to say the problem is transferred in this way from the real world to some idealistic world. The possibly infinitely many computations of the program remain as complex as before. On the other hand, it is by use of an operational semantics that the meaning of most of the existing programming languages is specified. Examples are the programming languages PL/1 in LUCAS & WALK 1971, and, underneath its special description method, ALGOL 68 in Van WIJNGAARDEN 1975.

For about 15 years so-called *denotational semantics* have been provided for programming languages (see e.g. TENNENT 1976, STOY 1977, De BAKKER 1980) of a program is given as a mathematical object in a model; usually some function which describes the input-output behaviour of the program. By abstracting from the intermediate stages of the computation, the model has far less resemblance to a real computer than the abstract machines used in operational semantics. The programs are not considered so much to be transforming values into values, but rather as transforming the entire initial state of a computer into some final state. In this approach, states are highly complex descriptions of all information present in the computer.

Mostly, we are not interested in all aspects of a computer state, but only in a small part (for instance the values of the input and output variables). This leads to a third approach to semantics, which uses so-called *predicate transformers* (FLOYD 1967, HOARE 1969, DIJKSTRA 1974, 1976 and DE BAKKER 1980). A (state) predicate is a proposition about states. So a predicate specifies a set of states: all states for which the proposition holds true. We need to correlate propositions about the state before the execution of the program (*preconditions*) with propositions about the state afterwards (*postconditions*). This is the approach to semantics that we will follow in the sequel. Usually one distinguishes approaches which associate preconditions and postconditions, but do not consider termination of the execution of the program, and approaches which consider termination as well. The former approaches are said to deal with *partial correctness*, and the latter with *total correctness*. Since all programs we will discuss are terminating programs, the distinction is for our fragment not relevant and will not be mentioned any further.

As an example we consider the program from Section 2.1. An initial state may be described by specifying that on the input channel three numbers a, b and c are present such that $a \neq 0$, and $b^2 - 4ac \geq 0$. The execution of the program will lead such a state to a state where $x1$ and $x2$ contain the solutions to the equation $ax^2 + bx + c = 0$. Conversely, we observe that, if one wants the program to stop in a state where $x1$ and $x2$ represent the solutions of the equation $ax^2 + bx + c = 0$, it suffices to require that the coefficients a, b and c are present on the input channel (in this order!) before the execution of the program, and that moreover $a \neq 0$ and $b^2 - 4ac \geq 0$. In the semantics we will restrict our attention to the real computation, and therefore consider a reduced version of the program from which the input and output instructions and the specifications of the identifiers such as <u>*real*</u> are removed. Let us call this reduced program 'prog'. In presenting the relation between predicates and programs, we follow a notational convention due to HOARE 1969. Let π be a program, and ϕ and ψ predicates expressing properties of states. Then $\{\phi\}\pi\{\psi\}$ means that if we execute π starting in a state where ϕ holds true, and the execution of the program terminates, then predicate ψ holds in the resulting state. Our observations concerning the program are now expressed by:

(9) $$\{\alpha \neq 0 \wedge (b^2-4ac) \geq 0\} \text{ prog } \{\alpha(x1)^2 + b(x1) + c = 0 \wedge a(x2)^2 + b(x2) + c = 0 \wedge \forall z[az^2 + bz + c = 0 \rightarrow z = x1 \vee z = x2]\}.$$

There are two variants of predicate transformer semantics. The aim of the first variant, the *forward approach* or (*Floyd-approach*) can be described as follows. For any program π, find, according to the structure of π, a predicate transformer which for any state predicate ϕ yields a state predicate ψ, such that if ϕ holds before the execution of π, then ψ gives all information about the final state which can be concluded from ϕ and π. Such a predicate ψ is called a *strongest postcondition* with respect to ϕ and π.

Mathematically a strongest postcondition sp (with respect to ϕ and π) is defined by

(I) $\{\phi\}\ \pi\ \{sp\}$ and

(II) If $\{\phi\}\ \pi\ \{\eta\}$ then from sp we can conclude η.

Suppose that we have two predicates sp_1 and sp_2, both satisfying (I) and (II). Then they are equivalent. From (I) follows that $\{\phi\}\ \pi\ \{sp_1\}$ and $\{\phi\}\ \pi\ \{sp_2\}$. Then from (II) follows that sp_1 implies sp_2 and vice versa. Since all strongest postcondition with respect to ϕ and π, are equivalent, we may speak about *the* strongest postcondition with respect to ϕ and π. For this the notation $sp(\pi,\phi)$ is used.

Instead of this approach, one frequently follows an approach which reverses the process: the *backward-approach* or *Hoare-approach*. For a program π and a predicate ψ one wants to find the weakest predicate which still ensures that, after execution of π, predicate ψ holds. Such a predicate is called a *weakest precondition*. Mathematically a weakest precondition wp (with respect to π and ψ) is defined by

I $\{wp\}\ \pi\ \{\psi\}$

II If $\{\eta\}\ \pi\ \{\psi\}$ then from η we can conclude wp.

Analogously to the proof for postconditions, it can be shown that all weakest preconditions are equivalent. Therefore we may speak about *the* weakest precondition with respect to π and ϕ. For this the notation $wp(\pi,\phi)$ is used (see DIJKSTRA 1974, 1976 for more on this approach).

Above, I used the phrase 'based upon the structure of π'. This was required since it would be useless to have a semantics which attaches to each program and predicate a strongest postcondition in an ad-hoc way, in particular because there are infinitely many programs. One has to use the fact that programs are formed in a structured way according to the syntax of the programming language, and according to our framework, we aim at obtaining these predicate transformers by means of a method which employs this structure.

3. PREDICATE TRANSFORMERS

3.1. Floyd's forward predicate transformer

Below, Floyd's description is given of the strongest postcondition for the assignment statement. But before doing so, I give some suggestive heuristics. Suppose that $x = 0$ holds before the execution of $x := 1$. Then afterwards $x = 1$ should hold instead of $x = 0$. As a first guess at a generalization one might suppose that always after execution of $v := \delta$ it holds that $v = \delta$. But this is not generally correct, as can be seen from inspection of the assignment $x := x + 1$. One must not confuse the old value of a variable with the new one. To capture this old value versus new-value distinction, the information about the old value is remembered using a variable (in the logical sense!) bound by some existential quantifier and using the operation of substitution. So after $v := \delta$ one should have that v equals 'δ with the old value of v substituted (where necessary) for v in δ'. This paraphrase is expressed by the expression $v = [z/v]\ \delta$, where z stands for the old value of v and $[z/v]$ is the substitution operator. Thus we have obtained information about the final situation from the assignment statement itself. Furthermore we can obtain information from the information we have about the situation before the execution of the assignment. Suppose that ϕ holds true before the execution of the assignment. From the discussion in Section 2 we know that the execution of $v := \delta$ changes only the value of v. All information in ϕ which is independent of v remains true. So after the execution of the assignment $[z/v]\phi$ holds true. If we combine these two sources of information into one formula, we obtain Floyd's *forward predicate transformation rule* for the assignment statement (FLOYD 1967).

(10) $\{\phi\}\ v{:=}\delta\ \{\exists z[[z/v]\phi \wedge v = [z/v]\delta]\}.$

Here ϕ denotes an assertion on the state of the computer, i.e., the values of the relevant variables in the program before execution of the assignment, and the more complex assertion $\exists z[[z/v]\phi \wedge v = [z/v]\delta]$ describes the situation afterwards.

The examples below illustrate how the assignment rule works in practice.

1) assignment: $x := 1$; precondition: $x = 0$
   obtained postcondition:
   $\exists z[[z/x](x{=}0) \wedge x = [z/x]1]$, i.e. $\exists z[z{=}0 \wedge x{=}1]$, which is equivalent to $x = 1$.
2) assignment: $x := x + 1$; precondition: $x > 0$
   obtained postcondition:
   $\exists z[[z/x](x{>}0) \wedge x = [z/x](x{+}1)]$, i.e. $\exists z[z{>}0 \wedge x{=}z{+}1]$, which is equivalent to $x > 1$.

3) Assignment: $a[1] := a[1] + 1$; precondition: $a[1] = a[2]$.
   Obtained postcondition:
   $\exists z[[z/a[1]](a[1] = a[2]) \wedge a[1] = [z/a[1]](a[1]+1)]$, i.e.
   $\exists z[z = a[2] \wedge a[1] = z+1]$, which is equivalent to $a[1] = a[2] + 1$.

3.2. Hoare's backward predicate transformer

Below Hoare's description will be given of the weakest precondition for the assignment statement. First I will give some heuristics. Suppose we want $x = 4$ to hold after the execution of $x := y + 1$. Then it has to be the case that before the execution of the assignment, $y + 1 = 4$ holds. More generally, every statement about x holding after the assignment has to be true about $y + 1$ before its execution. This observation is described in the following rule for the backward predicate transformer (HOARE 1969)

(11) $\{[\delta/v]\phi\}\ v := \delta\ \{\phi\}$.

Some examples illustrate how the rule works in practice.

1) Assignment: $x := 1$; postcondition: $x = 1$.
   Obtained precondition:
   $[1/x](x=1)$, i.e. $1 = 1$, or *true*.
   This result says that for *all* initial states $x = 1$ holds after the execution of the assignment. If the postcondition had been $x = 2$, the obtained precondition would have been $1 = 2$ or *false*, thus formalizing that for *no* initial state does $x = 2$ hold after execution of $x := 1$.
2) Assignment: $x := x + 1$; postcondition $x > 1$.
   Obtained precondition:
   $[x+1/x](x>1)$, i.e. $x + 1 > 1$ which is equivalent to $x > 0$.
3) Assignment: $a[1] := a[1] + 1$, postcondition $a[1] = a[2] + 1$.
   Obtained precondition:
   $[a[1] + 1/a[1]](a[1] = (a[2]+1))$, i.e. $a[1] + 1 = a[2] + 1$, which is equivalent with $a[1] = a[2]$.

3.3. Problems with Floyd's rule

Since 1974 it has been noticed by several authors that the assignment rules of Floyd and Hoare lead to incorrect results when applied to cases where the identifier is not directly associated with a cell storing an integer value. Examples are given in Van EMDE BOAS (1974), (thesis 13), De BAKKER (1976), GRIES (1977), JANSSEN & Van EMDE BOAS (1977a,b). The examples concern assignments involving an identifier of an integer array, or a pointer to an integer identifier. In this section I will consider only examples concerning Floyd's rule.

An example concerning assignment to a subscripted array identifier is

(12) $a[a[1]] := 2$.

Suppose that the assertion which holds before the execution of the assignment is

(13) $a[1] = 1 \wedge a[2] = 1$.

Then Floyd's rule implies that after the execution of the assignment holds

(14) $\exists z[[z/a[a[1]]](a[1]=1 \wedge a[2]=1) \wedge a[a[1]] = [z/a[a[1]]]2]$

i.e.

(15) $\exists z[a[1] = 1 \wedge a[2] = 1 \wedge a[a[1]] = 2]$

which is equivalent to

(16) $a[1] = 1 \wedge a[2] = 1 \wedge a[a[1]] = 2$.

This formula is a contradiction, whereas the assignment is a correctly terminating action. Compare this result with the situations in figure 3, where this assignment is performed in a situation satisfying the given precondition. Then it is clear that the postcondition should be

(17) $a[1] = 2 \wedge a[2] = 1$.

It turns out that problems also arise in the case of pointers (JANSSEN & Van EMDE BOAS 1977a). An example is the following program consisting of three consecutive assignment statements. The identifier p is a pointer and x an integer variable.

(18) $x := 5;\ p := x;\ x := 6$.

Suppose that we have no information about the state before the execution of this program. This can be expressed by saying that the predicate <u>*true*</u> holds in the initial state. By application of Floyd's rule, we find that after the first assignment $x = 5$ holds (analogously to the first example above). Note that the state presented in figure 2a (Section 1) satisfies this predicate. For the state after the second assignment Floyd's rule yields:

(19) $\exists z[[z/p](x=5) \wedge p = [z/p]x]$

i.e.

(20) $\exists z[x{=}5 \wedge p{=}x]$

which is equivalent to

(21) $x = 5 \wedge p = x$.

It is indeed the case that after the second assignment the integer value related with p equals 5 (of figure 2b). According to Floyd's rule, after the third assignment the following is true:

(22) $\exists z[[z/x](x{=}5 \wedge p{=}x) \wedge x = [z/x]6]$

i.e.

(23) $\exists z[z = 5 \wedge p = z \wedge x = 6]$.

This formula says that the integer value related with p equals 5. But as the reader may remember from the discussion in Section 2, the integer value related with p is changed as well (figure 2c).

3.4. Predicate transformers as meanings

Floyd's assignment rule is one rule from a collection of proof rules: for each construction of the programming language there is a rule which describes a relation between precondition and post condition. The meaning of a construction is defined in a completely different way. A computer-like model is defined, and the meaning of a statement (e.g. the assignment statement) is described as a certain state-transition function (a function from computer states to computer states). The proof rule corresponding to the construction can be used to prove properties of programs containing this construction. A prime example of this approach is De BAKKER (1980). It is, however, not precisely the approach that I will follow in this chapter.

In the discussion in section 2.2 I have mentioned arguments why predicate transformers are attractive from a semantic viewpoint, and why state-transition function are less attractive. I will give predicate transformers a central position in my treatment: the meaning of a program, and in particular of an assignment statement, will be *defined* by means of a predicate transformer.

In theory I could define the meaning of an assignment by any predicate transformer I would like. But then there is a great danger of loosing contact with the behaviour of computer programs in practice. Therefore I will give a justification of my choice of the predicate transformers. This will be done by defining a state-transition function that resembles the usual state-transition semantics. Then it will be proven that the defined predicate transformers are correct and yield strongest postconditions (or weakest preconditions). In the light of this connection with practice, it is not surprising that there is a resemblence between Floyd's (Hoare's) predicate transformer and the one I will define. But the formal position of the predicate transformers is essentially different in this approach. Actually, I shall argue that Floyd's (Hoare's) predicate transformer cannot be used for our purposes. The problems with the standard formulation of the transformers are mentioned below; they are solvable by some modifications which will be discussed in the next section. The discussion will be restricted to the Floyd-approach; for the Hoare approach similar remarks apply.

In the Floyd-approach the predicate-transformation rule for the assignment is an axiom in a system of proof rules. It can be considered as an instruction how to change a given predicate into its strongest postcondition. In our approach an assignment statement has to be considered semantically as a predicate transformer. Hence it has to correspond with a single expression which is interpreted in the model as a predicate transformer. This requires that Floyd's rule has to be reformulated into such an expression. This can be done by means of a suitable λ-abstraction. The predicate transformer corresponding with assignment $x := \delta$ will look like (24).

(24) $\lambda\phi\exists z[[z/x]\phi \wedge x = [z/x]\delta]$.

This expression is not quite correct because of an inconsistency in the types of ϕ. The subexpression $[z/x]\phi$ is part of a conjunction. Therefore both $[z/x]\phi$ and ϕ have to denote a truth-value. But in the abstraction $\lambda\phi$ the ϕ is not intended as an abstraction over truth-values (there are only two of them), but as an abstraction over predicates (there are a lot of them). This means that the types of ϕ in (24) are not consistent, so it cannot be the predicate-transformer which we will use.

A second problem is the occurrence of the substitution operator in Floyd's rule (and in (24)). It is an operator which operates on strings of symbols. The operator does not belong to the language of logic and there is no semantic interpretation for it. Hence expressions containing the operator have no interpretation. To say it in the terminology of our framework: expressions like (24) are not a polynomial operator over the logic used. Remember that no logical language has the substitution operator as one of its operators. Substitution belongs to the meta-language, and is used there to indicate how an expression of the logic has to be changed in order to obtain a certain other expression. Since proof rules and axioms are, by their nature, rules concerning syntactic objects, there is no objection against a substitution operator occurring in a proof rule. But we wish to use predicate transformers to determine meanings. If we would use substitution operators in predicate-transformers, then our transformers would be instructions for formula manipulation, and we would not do semantics. The same observation is made by Tennent with respect to another rule. He stated in a discussion (NEUHOLD 1978, p.69):

Substitution is purely syntactic, function modification semantic.

The third problem can be illustrated by considering the assignment $x := y + 1$. The identifier x is used in the execution of the program in an essentially different way than the identifier y. The y is used to indicate a certain value. The x is used as the name of a cell, and not to indicate a value. This different use corresponds with the semantic difference : in section 1.2 we observed that the left-hand side of the assignment statement is referentially opaque, whereas the right-hand side is transparent. Floyd's rule does not reflect these differences. The rule makes no clear distinction between a name and the value associated with that name. In my opinion this is the main source of the problems with Floyd's rule. Remember that all problems we considered above, arose precisely in those situations where there are several ways available for referring to a certain value in the computer: one may use an identifier or a pointer to that identifier; one may use an array identifier subscripted with an integer, or subscripted with an compound expression referring to the same value.

In the field of semantics of natural languages an approach which identified name and object-referred-to was employed in the beginnings of this century. Ryle epitomizes this feature of these theories in his name for them: 'Fido'-Fido theories! The word 'Fido' means Fido, the dog, which is its meaning (see STEINBERG & JAKOBOVITS 1971, p.7). The approach was abandoned, because it turned out to be too simple for treating the less elementary cases. In view of the analogy of the behaviour of names in natural languages and in programming languages we observed in section 1, it is not too surprising that Floyd's rule is not completely successful either.

4. SEMANTICAL CONSIDERATIONS

4.1. The model

In section 5 the syntax and semantics of a small fragment of a programming language will be presented; in section 7 a larger fragment will be dealt with. The treatment will fit the framework developed in the first chapter. So we will translate the programming language into some logical language, which is interpreted in some model. In the present section the semantical aspects (model, logic) will be discussed which are relevant for the treatment of the first fragment. In sections 6 and 7 this discussion will be continued.

In section 2.1 we observed that the assignment statement creates an intensional context. Therefore it is tempting to try to apply in the field of programming languages the notions developed for intensional phenomena in natural languages. The basic step for such an application is the transfer of the notion 'possible world' to the context of programming languages. It turns out that possible worlds can be interpreted as internal states of the computer. Since this is a rather concrete interpretation, I expect that the ontological objections which are sometimes raised against the use of possible world semantics for natural languages (e.g. POTTS 1976), do not apply here. The idea to use a possible world semantics and some kind of modal logic can be found with several authors. An influencing article in this direction was PRATT 1976; for a survey, see Van EMDE BOAS 1978 or PRATT 1980.

An important set in the model is the set of possible worlds, which in the present context will be called set of states. This set will be introduced in the same way as possible worlds were introduced in the treatment of natural languages. It is just some non-empty set (denoted by ST). They are not further analysed; so we do not build explicitly in our semantic domains some abstract model of the computer. But this does not mean that every model for intensional logic is an acceptable candidate for the interpretation of programming languages. Below I will formulate some restrictions on these models, which determine a certain subclass, and these restrictions have, of course consequences for the set ST as well. In this indirect way certain properties of the computer are incorporated in the model. The formulation of the restrictions only concern the simple assignment statement, and they will be generalized in section 7.

An integer identifier is associated with some cell in the computer, and for each state we may ask which value is contained in this cell. The semantic property of an integer identifier we are interested in, is the function which relates a state with the value contained (in that state) in the cell corresponding to that identifier. So we wish to associate with an identifier a function from states to values, see chapter 1 for a discussion (the same idea can be found in ADJ 1977 or 1979). In order to obtain this effect, integer identifiers are translated into constants of type <s,e> (e.g. the identifiers *x*,*y* and *w* are translated into the constants *x*,*y* and *w* of type <s,e>). But something more can be said about their interpretation. The standard interpretation of constants of intensional logic allows that for a given constant we obtain for different states different functions from states to values as interpretation. But we assume that on the computers on which the programs are executed, the relation between an identifier and the corresponding cell is never changed, so that for all states the function associated with an identifier is the same. The interpretations of *x*,*y* and *w* have to be state independent (in chapter 5, section 2 a related situation will arise for natural language; one uses there for such constants the name 'rigid designators'). This requirement implies that not all models for intensional logic are acceptable as candidates for formalizing the meaning of programming languages. We are only interested in those models in which the following postulate holds.

4.1. Rigidness Postulate

Let $c \in \mathrm{CON}_{<s,e>}$ and $v \in \mathrm{VAR}_{<s,e>}$. Then the following formula holds:

$$\exists v \Box [c=v].$$

4.1. END

The above argumentation in favour of the rigidness postulate is not completely compelling. For a fragment containing only simple assignment statements one might alternatively translate integer identifiers into constants of type *e* which are interpreted non-rigidly. In such an approach the constant relation between an identifier and a cell would not have been formalized. This aspect will, however, become essentail if the fragment is extended with pointers. Although there are no essentially non-rigid constants in the fragment under consideration, it is also possible to consider such constructs e.g. the integer identifier *xory* which denotes the same as the integer identifier x or the integer identifier y, depending on which of both currently has the greatest integer value. The rigidness postulate guarantees that the interpretation of constants is state independent. Therefore we may replace the usual notation for their interpretation, being $F(c)(s)$, by some notation not mentioning the current state. I will use $V(c)$ as the notation for the interpretation of a constant with respect to an arbitrary state.

Two states which agree in the values of all identifiers should not be distinguishable, since on a real computer such states (should) behave alike. Two states only count as different if they are different with respect to the value of at least one identifier. This is expressed in the following postulate.

4.2. Distinctness Postulate

Let $s,t \in ST$. If for all $c \in CON$, $V(c)(s) = V(c)(t)$, then $s = t$.

4.2. END

The execution of an assignment modifies the state of the computer in a specific way: the value of a single identifier is changed, while the values of all other identifiers are kept intact. This property is expressed by the update postulate, which requires that the model to be rich enough to allow for such a change. The term 'update' should not be interpreted as stating that we change the model in some way; the model is required to have a structure allowing for such a transition of states.

4.3. Update Postulate

For all $s \in ST$, $c \in CON_{<s,e>}$, $n \in \mathbb{N}$ there is a $t \in ST$ such that

$$V(c)(t) = n$$
$$V(c')(t) = V(c')(s) \quad \text{if} \quad c' \neq c.$$

4.3. END

The update postulate requires the existence of a certain new state, and the distinctness postulate guarantees the uniqueness of this new state.

I formulated the update postulate for constants of type <s,e> only, but in section 7 it will be generalized to constants of many other types as well. If the update postulate holds for a constant c and a value d, then the (unique) state required by the postulate is denoted $<c \leftarrow d>s$.

Note that the postulates differ from the meaning postulates given for natural languages in the sense that they are formulated in the meta-language and not in intensional logic itself. This allowed us to use quantification over states and over constants in the formulation of the postulates.

One might wish to construct a model which satisfies these three postulates. It turns out that the easiest way is to give the states an internal structure. The rigidness postulate and the distinctness postulate say that we may take for elements of ST sets of functions from (translations of) identifiers to integers. The update postulate says that ST has to be a sufficiently large set. Let ID be the set of integer identifiers. Then we might take $ST = \mathbb{N}^{ID}$. Another possibility (suggested by J. Zucker) is $ST = \{s \in \mathbb{N}^{ID} \mid s(x) \neq 0 \text{ for only finitely many } x\}$. Sets of states with a completely different structure are, in principle, possible as well.

In the introduction I have said that the set of states (set of possible worlds) is just some set. This means that states are, in our approach, a primitive notion and that no internal structure is required for them. But the models just described correspond closely with the models know from the literature (e.g. the one defined by De BAKKER (1980, p.21)); for the larger fragment we will consider this correspondence is less obvious (see section 7). The difference between these two approaches is that here we started with requiring certain properties, whereas usually one starts defining a model. A consequence is that we are only allowed to use the properties we explicitly required, and that we are not allowed to use the accidental properties of a particular model. This is an advantage when a model has to be explicit about a certain aspect, whereas a theory is required to be neutral in this respect. An example could be the way of initialization of identifiers as discussed in De BAKKER (1980, p.218). He says about a certain kind of examples that it: '[..] indicates an overspecification in our semantics [..], it also leads to an incomplete proof theory'. He avoids the problem by eliminating them from his fragment. By means of the present approach such an overspecification could probably avoided.

4.2. The logic

We will use a possible-world semantics for dealing with phenomena of opaque and transparant contexts. Therefore it is tempting to use as logical language the same language as we used in the previous chapters: intensional logic. Since we deal with a programming language, some of the semantic phenomena will differ considerably from the ones we considered before. Intensional logic will be extended with some new operators which allow us to cope with these new phenomena.

The programs deal with numbers, and this induces some changes. The constants of type e ($v_{1,e}, v_{2,e}, \ldots$) will be written in the form *0,1,2,3* .. and interpreted as the corresponding numbers. The logic is extended with operators on numbers: +, x, −, ≤, ≥, =. The symbols *true* and *false* abbreviate *1* = *1* and *1* ≠ *1* respectively. The programming language has an *if-then-else-fi* construction (the *fi* plays the role of a closing bracket; it eliminates syntactic ambiguities). A related construction is introduced in the logic. Its syntax and semantics are as follows:

4.4. DEFINITION. For all $\tau \in Ty$, $\alpha \in ME_t$, $\beta \in ME_\tau$ and $\gamma \in ME_\tau$ we have

$$\underline{if}\ \alpha\ \underline{then}\ \beta\ \underline{else}\ \gamma\ \underline{fi} \in ME_\tau .$$

The interpretation is defined by:

$$V_{s,g}\ \underline{if}\ \alpha\ \underline{then}\ \beta\ \underline{else}\ \gamma\ \underline{fi} = \begin{cases} V_{s,g}(\beta) & \text{if } V_{s,g}(\alpha) = 1 \\ V_{s,g}(\gamma) & \text{otherwise.} \end{cases}$$

4.4. END

The update postulate and the distinctness postulate guarantee for $n \in \mathbb{N}$ and $c \in CON_e$ existence and uniqueness of a state $<c{\leftarrow}n>s$. It is useful to have in the logic an operator which corresponds with the semantic operator $<c{\leftarrow}n>$. These operators, which I will call *state switchers*, are modal operators (since they change the state, (i.e. world) with respect to which its argument is interpreted). The syntax and semantics of state switchers is defined as follows.

4.5. DEFINITION. For all $\sigma,\tau \in CAT$, $\phi \in ME_\sigma$, $c \in CON_{<s,\tau>}$, $\alpha \in ME_\tau$ we have

$\{\alpha/{}^{\vee}c\}\phi \in ME_{\sigma}$.

The interpretation is defined by:

$$V_{s,g}(\{\alpha/{}^{\vee}c\}\phi) = \begin{cases} V_{<c\leftarrow V_{s,g}(\alpha)>s,g}(\phi) & \text{if } <c\leftarrow V_{s,g}(\alpha)>s \\ & \text{is defined,} \\ V_{s,g}(\phi) & \text{otherwise.} \end{cases}$$

Note that in the present stage of exposition, the 'defined' case only applies for $c \in CON_{<s,e>}$.
4.5. END

One might wonder why the state-switcher contains an extension operator, for only the constant c and the expression α are relevant for determing which state-switcher is intended. The reason is that state-switchers have many properties in common with the well-known substitution operators. The state-switcher determined by c and α behaves almost the same as the substitution operator $[\alpha/{}^{\vee}c]$. This will be proven in section 4.3.

The meaning of a program will be defined as a predicate transformer. Since we will represent meanings in intensional logic, we have to find a representation of predicate transformers in intensional logic. Let us first consider state-predicates. These are properties of states. For some states the predicate holds, for others it does not hold, so a state predicate is a function $f\colon S \rightarrow \{0,1\}$. Since the interpretation of intensional logic is state-dependent, such a state predicate can be represented by means of an expression of type t.

A predicate transformer should, in the present approach, not be an operation on expressions, but a semantic function which relates state-predicates with state-predicates. So it should be a function $f\colon (S\rightarrow\{0,1\}) \rightarrow (S\rightarrow\{0,1\})$. This means that it is a function which yields a truth-value, and which takes two arguments: a state-predicate, and a state. Changing the order of the arguments does not change the function essentially. We may consider a state-predicate as a function which takes a state and a state-predicate, and yields a truth-value. Hence we may say that a predicate transformer is a function $f\colon S \rightarrow ((S\rightarrow\{0,1\}) \rightarrow \{0,1\})$. This view is, in a certain sense, equivalent to the one we started with. A formula of type <<s,t>,s> has as its meaning such a function, hence formulas of type

$<<s,t>,t>$ are suitable as representations of predicate transformers. Therefore programs and assignments can be translated into expressions of this type.

One might have expected that programs and assignments are translated into expressions of type $<<s,t>,<s,t>>$. This was the type of the translations of programs and assignments in JANSSEN & Van EMDE BOAS (1977a,b). The first argument for using the type $<<s,t>,t>$ of theoretical nature. An expression of type $<<s,t>,<s,t>>$ has as its meaning a function $f\colon S \rightarrow ((S\rightarrow\{0,1\}) \rightarrow (S\rightarrow\{0,1\}))$, and this is not a predicate transformer (although it is closely connected, and could be used for that purpose). The second argument is of practical nature: the type of the present translation gives rise to less occurrences of the $^\wedge$ and $^\vee$ signs.

A consequence of the representations which we use for (state-)predicates and predicate transformers is the following. Suppose that program π is translated into predicate transformer π', and that this program is executed in a state which satisfies predicate ϕ. Then in the resulting state the predicate denoted by $\pi'(^\wedge\phi)$ holds; it is intended as the strongest condition with respect to program π and predicate ϕ (i.e. $sp(\pi,\phi)$).

4.3. Theorems

The substitution theorem says that the state-switcher behaves almost the same as the ordinary substitution operator. The iteration theorem describes a property of the iteration of state-switchers.

4.6. SUBSTITUTION THEOREM. *The following equalities hold with respect to all variable assignments and states.*

1. $\{a/^\vee c\}c' = c'$ *for all* $c' \in$ CON.
2. $\{\alpha/^\vee c\}v = v$ *for all* $v \in$ VAR.
3. $\{\alpha/^\vee c\}(\phi\wedge\psi) = \{\alpha/^\vee c\}\phi \wedge \{\alpha/^\vee c\}\psi$
   analogously for $\vee,\rightarrow,\leftrightarrow,\neg$, *if-then-else-fi constructs.*
4. $\{\alpha/^\vee c\}(\exists x\phi) = \exists x\{\alpha/^\vee c\}\phi$ *if x does not occur free in* α
   analogously for $\forall x\phi$, $\lambda x\phi$.
5. $\{\alpha/^\vee c\}(\beta(\gamma)) = [\{\alpha/^\vee c\}\beta](\{\alpha/^\vee c\}\gamma)$.
6. $\{\alpha/^\vee c\}^\wedge\beta = {}^\wedge\beta$ *analogously for* $\Box\beta$.
7. $\{\alpha/^\vee c\}^\vee c = \alpha$.

Consequence

The state switcher $\{\alpha/{}^{\vee}c\}$ behaves as the substitution operator $[\alpha/{}^{\vee}c]$, except if applied to ${}^{\wedge}\beta, \Box\beta$ or ${}^{\vee}\beta$ (where $\beta \not\equiv c$). The formulas $\{\alpha/{}^{\vee}c\}{}^{\wedge}\beta$ and $\{\alpha/{}^{\vee}c\}\Box\beta$ reduce to ${}^{\wedge}\beta$ and $\Box\beta$ respectively, whereas $\{\alpha/{}^{\vee}c\}{}^{\vee}\beta$ cannot be reduced any further.

PROOF. Let t be the state $<c{\leftarrow}V_{s,g}(\alpha)>s$, so $V_{s,g}\{\alpha/{}^{\vee}c\}\phi = V_{t,g}\phi$.

1. $V_{s,g}(\{\alpha/{}^{\vee}c\}c') = V_{t,g}(c') = V_{s,g}(c')$.

   The equalities hold because of the Rigidness Postulate.

2. $V_{s,g}\ \{\alpha/{}^{\vee}c\}(v) = V_{t,g}(v) = g(v) = V_{s,g}(v)$.

3. $V_{s,g}\{\alpha/{}^{\vee}c\}(\phi\wedge\psi) = 1 \iff V_{t,g}(\phi\wedge\psi) = 1 \iff V_{t,g}(\phi) = 1$ and $V_{t,g}(\psi) = 1 \iff V_{s,g}(\{\alpha/{}^{\vee}c\}\phi) = 1$ and $V_{s,g}(\{\alpha/{}^{\vee}c\}\psi) = 1 \iff V_{s,g}(\{\alpha/{}^{\vee}c\}\phi \wedge \{\alpha/{}^{\vee}c\}\psi) = 1$.

   Analogously for the other connectives.

4. $V_{s,g}(\{\alpha/{}^{\vee}c\}\exists x\phi) = 1 \iff V_{t,g}(\exists x\phi) = 1 \iff$ there is a $g' \sim_x g$ such that $V_{t,g'}(\phi) = 1 \iff \{x$ not free in $\alpha!\} \iff$ there is a $g' \sim_x g$ such that $V_{s,g}(\{\alpha/{}^{\vee}c\}\phi) = 1 \iff V_{s,g}(\exists x\{\alpha/{}^{\vee}c\}\phi) = 1$.

5. $V_{s,g}\{\alpha/{}^{\vee}c\}(\beta(\gamma)) = V_{t,g}(\beta(\gamma)) = V_{t,g}(\beta)(V_{t,g}(\gamma)) = V_{s,g}(\{\alpha/{}^{\vee}c\}\beta)(V_{s,g}\{\alpha/{}^{\vee}c\}\gamma) = V_{s,g}(\{\alpha/{}^{\vee}c\}\beta(\{\alpha/{}^{\vee}c\}\gamma))$.

6. $V_{s,g}(\{\alpha/{}^{\vee}c\}{}^{\wedge}\beta) = V_{t,g}({}^{\wedge}\beta) = \underline{\lambda} t'\ V_{t',g}(\beta) = V_{s,g}({}^{\wedge}\beta)$.

7. $V_{s,g}(\{\alpha/{}^{\vee}c\}{}^{\vee}c) = V_{t,g}({}^{\vee}c) = V(c)(<c{\leftarrow}V_{s,g}(\alpha)>s) = V_{s,g}(\alpha)$.

4.7. ITERATION THEOREM.

$$\{\alpha_1/{}^{\vee}c\}(\{\alpha_2/{}^{\vee}c\}\phi) = \{\{\alpha_1/{}^{\vee}c\}\alpha_2/{}^{\vee}c\}(\phi).$$

PROOF. Note that also here the state switcher behaves as a substitution operator: first a substitution of α_2 for all occurrences of ${}^{\vee}c$, and next a substitution of α_1 for the new occurrences of ${}^{\vee}c$, is equivalent with an immedaite substitution of $[\alpha_1/{}^{\vee}c]\alpha_2$ for all occurrences of ${}^{\vee}c$. The proof of the theorem is as follows.

First consider $<c{\leftarrow}d_1>(<c{\leftarrow}d_2>)s$, where d_1 and d_2 are possible values of c. This denotes a state in which all identifiers have the same value as in s, except for c which has value d_1. So it is the same state as $<c{\leftarrow}d_1>s$

(due to the distinctness postulate). This equivalence is used in the proof below.

$$V_{s,g}\{\alpha_1/{}^{\vee}c\}(\{\alpha_2/{}^{\vee}c\}\phi) = V_{<c\leftarrow V_{s,g}(\alpha_1)>s,g}\{\alpha_2/{}^{\vee}c\}(\phi) =$$

$$V_{<c\leftarrow V_{<c\leftarrow V_{s,g}(\alpha_1)>s,g}(\alpha_2)>(<c\leftarrow V_{s,g}(\alpha_1)>)s,g}(\phi) =$$

$$V_{<c\leftarrow V_{<c\leftarrow V_{s,g}(\alpha_1)>s,g}(\alpha_2)>s,g}(\phi) =$$

$$V_{<c\leftarrow V_{s,g}(\{\alpha_1/{}^{\vee}c\}\alpha_2)>s,g}(\phi) = V_{s,g}\{\{\alpha_1/{}^{\vee}c\}\alpha_2/{}^{\vee}c\}(\phi)$$

4.7. END

5. FIRST FRAGMENT

5.1. The rules

In this section the syntax and semantics will be presented of a small fragment of a programming language. The fragment contains only programs which consist of a sequence of simple assignment statements; many programming languages have a fragment like the one presented here. The treatment will be in accordance with the framework developed in the first chapters of this book. This means that for each basic expression (generator of the syntactic algebra) there has to be a translation into the logic, and that for each syntactic rule there has to be a corresponding semantic rule which says how the translations of the parts of a syntactic construction have to be combined in order to obtain the meaning of the compound construction.

The syntax of the fragment has the following five categories:

1.	INT	The set of representations of integers. Basic expressions in this category are: *1,2,3,...,12,...,666,...* .
2.	ID	The set of integer identifiers. Basic expressions are *x,y* and *z*.
3.	ASS	The set of assignments.
4.	PROG	The set of programs.
5.	BOOL	The set of boolean expressions.

The basic expressions of the category INT translate into corresponding constants of type e; the translation of *1* is 1 etc. The identifiers *x,y* and *w* translate into corresponding constants of type <s,e>: the translation of *x* is x.

The syntactic rules are presented in the same way as in previous chapters. In the clause called 'rule', the categories involved are mentioned;

first the categories of the input expressions, then the category of the resulting expression. The F-clause describes the operation which is performed on the input expressions; here α always stands for the first input expression, β for the second, and γ for the third. The T-clause describes how the translation of the resulting expression is built up from the translations of the input expressions. Here α' denotes the translation of the first input expression, β' of the second, and γ' of the third.

Rule S_{1a}: INT $\times$ INT $\rightarrow$ BOOL
F_{1a}: $\alpha = \beta$
T_{1a}: $\alpha' = \beta'$.

Example S_{1a}: Out of the integer expressions *1* and *2*, we may build the boolean expression *1 < 2*, with as translation *1 < 2*.

Rules $S_{1b}..S_{1e}$: Analogously for the relations $>, \leq, \geq, =$.

Rule S_{2a}: INT $\times$ INT $\rightarrow$ INT
F_{2a}: $\alpha + \beta$
T_{2a}: $\alpha' + \beta'$

Example : $(1+2)' = 1 + 2$

Rules S_{2b}, S_{2c}: Analogously for the operations $\times$ and $\div$

Rule S_3 : ID $\rightarrow$ INT
F_3 : α
T_3 : ${}^{\vee}\alpha'$

Example : The integer identifier *x* can be used to denote an integer.

Rule S_4 : ID $\times$ INT $\rightarrow$ ASS
F_4 : $\alpha := \beta$
T_4 : $\lambda P[\exists z[\{z/{}^{\vee}\alpha'\}{}^{\vee}P \wedge {}^{\vee}\alpha' = \{z/{}^{\vee}\alpha'\}\beta']]$ $(z \in \mathrm{VAR}_e)$

Example : See below. Notice the similarity and differences between this predicate transformer and Floyd's original rule. Some extension operators have been added, and the substitution operator is replaced by an operator with a semantical interpretation.

Rule S_5 : ASS $\rightarrow$ PROG
F_5 : α
T_5 : α'

Example : Every assignment statement can be used as a (reduced) program.

Rule S_6 : PROG × PROG → PROG
F_6 : $\alpha;\beta$
T_6 : $\lambda P[\alpha'(^\wedge\beta'(P))]$.

Rule S_7 : BOOL × PROG × PROG → PROG
F_7 : *if* α *then* β *else* γ *fi*
T_7 : $\lambda P[\beta'{}^\wedge(\alpha' \wedge {}^\vee P) \vee \gamma'{}^\wedge(\neg\alpha' \wedge {}^\vee P)]$

5.2. Examples

5.1. EXAMPLE: $x := y$.

The derivational history of this assignment is presented in figure 4. Also the successive steps of the translation process are presented in the tree. At each stage the number of the rule used and the category of the produced expression are mentioned between braces.

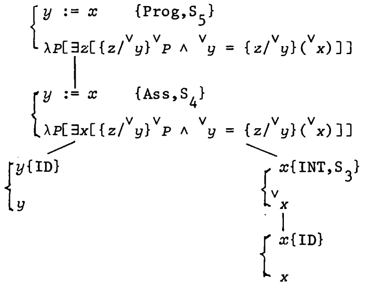

$y := x$ {Prog,S_5}
$\lambda P[\exists z[\{z/{}^\vee y\}{}^\vee P \wedge {}^\vee y = \{z/{}^\vee y\}({}^\vee x)]]$
$y := x$ {Ass,S_4}
$\lambda P[\exists x[\{z/{}^\vee y\}{}^\vee P \wedge {}^\vee y = \{z/{}^\vee y\}({}^\vee x)]]$
y\{ID\}
y
x\{INT,S_3\}
${}^\vee x$
x\{ID\}
x

Figure 4. $y := x$

The obtained translation of the program can be reduced, using the substitution theorem, to (25)

(25) $\lambda P[\exists z[\{z/{}^\vee y\}{}^\vee P \wedge {}^\vee y = {}^\vee x]]$.

Now suppose that before the execution of the assignment x equals 7 and y equals 2 (cf. Section 1, Figure 2c). So the initial state satisfies predicate (26):

(26) ${}^\vee x = 7 \wedge {}^\vee y = 2$.

Then after the execution of the assignment the following holds:

(27) $\lambda P[\exists z[\{z/{}^{\vee}y\}{}^{\vee}P \wedge {}^{\vee}y={}^{\vee}x]]({}^{\wedge}[{}^{\vee}x=7 \wedge {}^{\vee}y=2])$.

This reduces to (28), and further to (29) and (30).

(28) $\exists z[\{z/{}^{\vee}y\}({}^{\vee}x=7 \wedge {}^{\vee}y=2) \wedge {}^{\vee}y={}^{\vee}x]$

(29) $\exists z[{}^{\vee}x=7 \wedge z=2 \wedge {}^{\vee}y={}^{\vee}x]$

(30) ${}^{\vee}x=7 \wedge {}^{\vee}y={}^{\vee}x$.

5.2. EXAMPLE: $y := x$; $y := y + 1$.

The translation of the second assignment statement is obtained in the same way as the translation of $y := x$ in example 5.1. Its translation is (31), which reduces to (32).

(31) $\lambda P\exists z[\{z/{}^{\vee}y\}{}^{\vee}P \wedge {}^{\vee}y = \{z/{}^{\vee}y\}({}^{\vee}y+1)]$

(32) $\lambda P\exists z[\{z/{}^{\vee}y\}{}^{\vee}P \wedge {}^{\vee}y = z+1]$.

The translation of the whole program is therefore

$$\lambda Q[y:=y+1]'({}^{\wedge}[[y:=x]'(Q)]) =$$

$$\lambda Q\lambda P[\exists z[\{z/{}^{\vee}y\}{}^{\vee}P \wedge {}^{\vee}y = z+1]]({}^{\wedge}\exists z[\{z/{}^{\vee}y\}{}^{\vee}Q \wedge {}^{\vee}y={}^{\vee}x]) =$$

$$\lambda Q\exists z[\{z/{}^{\vee}y\}(\exists w[\{w/{}^{\vee}y\}{}^{\vee}Q \wedge {}^{\vee}y={}^{\vee}x]) \wedge {}^{\vee}y = z+1] =$$

$$\lambda Q\exists z\exists w[\{\{z/{}^{\vee}y\}w/{}^{\vee}y\}{}^{\vee}Q \wedge z={}^{\vee}x \wedge {}^{\vee}y = z+1] =$$

$$\lambda Q\exists z\exists w[\{w/{}^{\vee}y\}{}^{\vee}Q \wedge z={}^{\vee}x \wedge {}^{\vee}y = z+1].$$

Suppose now that before the execution of the program $x > 0$ holds. Then afterwards (33) holds, which reduces in turn to (34) and further to (35).

(33) $\exists z\exists w[\{w/{}^{\vee}y\}({}^{\vee}x>0) \wedge z={}^{\vee}x \wedge {}^{\vee}y=z+1]$

(34) $\exists z[{}^{\vee}x>0 \wedge z={}^{\vee}x \wedge {}^{\vee}y=z+1]$

(35) ${}^{\vee}x>0 \wedge {}^{\vee}y={}^{\vee}x+1$.

In the treatment of this program we first determined the translation of the program, and then considered some specific precondition. If we knew the precondition beforehand, and were only interested in obtaining the postcondition (and not in obtaining the translation of the whole program), we could first calculate the postcondition after the first assignment. This postcondition could then be taken as precondition for the second assignment.

5.3. EXAMPLE: *if* $y < 0$ *then* $y := x$ *else* $y := y+1$ *fi*.

The predicate transformer corresponding with this program is

(36) $\lambda Q[\lambda P[\exists z\{z/{}^{\vee}y\}{}^{\vee}P \wedge {}^{\vee}y={}^{\vee}x]^{\wedge}({}^{\vee}y<0 \wedge {}^{\vee}Q) \vee \lambda P[\exists z\{z/{}^{\vee}y\}{}^{\vee}P \wedge$

$$\wedge\ {}^{\vee}y=z+1]^{\wedge}(\neg[{}^{\vee}y<0] \wedge {}^{\vee}Q)].$$

This reduces to

(37) $\lambda Q[\exists z[z<0 \wedge \{z/{}^{\vee}y\}{}^{\vee}Q \wedge {}^{\vee}y={}^{\vee}x] \vee \exists z[\neg[z<0]\wedge\{z/{}^{\vee}y\}{}^{\vee}Q \wedge {}^{\vee}y=z+1]$.

Suppose that we have no information about the state before the execution of the assignment. This is expressed by the precondition $1=1$. Then afterwards (38) holds, which reduces to (39).

(38) $\exists z[z<0 \wedge {}^{\vee}y={}^{\vee}x] \vee \exists z[\neg[z<0] \wedge {}^{\vee}y=z+1]$

(39) ${}^{\vee}y={}^{\vee}x \vee {}^{\vee}y \geq 1$.

5.3. END

6. POINTERS AND ARRAYS

6.1. Pointers

An application of Floyd's rule to assignments containing pointers may give rise to problems, see the example in section 3. In sections 4-6 we have developed a compositional, algebraic approach for simple assignments. This algebraic approach can be generalized in a straightforward way to the case of pointers. I will consider at this moment only pointers to integer identifiers; a more general and formal treatment will be given in section 7.

Pointers to integer identifiers are expressions which have as value in a given state some integer identifier. In another state they may have another identifier as value. Therefore we associate with a pointer some function from states to interpretations of integer identifiers. In analogy to the treatment of integer identifiers, this is done by translating the pointer into a rigid constant; so pointer p translates into constants $p \in \mathrm{CON}_{<s,<s,e>>}$. The execution of the assignment $p := y$ has as an effect that the current state is changed in such a way that in the new state all identifiers have the same value as before, except for p which now has value y. This effect can be described by means of a state switcher like the ones we introduced in relation with simple assignments. Below I will introduce some postulates which guarantee that the state switchers $\{z/{}^{\vee}p\}$ can be interpreted in the way we intend, and satisfies equalities analogous to the substitution theorem (4.6). But first an example. We assume that the predicate transformer corresponding with the assignment $p := \delta$ reads:

(40) $\lambda P[\exists z[\{z/{}^{\vee}p\}{}^{\vee}P \wedge {}^{\vee}p = \{z/{}^{\vee}P\}\delta]]$ $\quad (z \in \mathrm{VAR}_{<s,e>})$.

6.1. EXAMPLE. $x := 5;\ p := x;\ x := 6$.
Let us assume that this program is executed in an arbitrary state, so the precondition is _true_. We are interested in the postcondition after the last assignment. That postcondition is obtained by calculating the postcondition of each assignment in turn, and taking that postcondition as input for the predicate transformer of the next assignment. The postcondition of the first assignment for precondition _true_ reduces as follows.

$$\lambda P[\exists z[\{z/{}^{\vee}x\}{}^{\vee}P \wedge {}^{\vee}x{=}5]]({}^{\wedge}\underline{true}) = \exists z[{}^{\vee}x{=}5] = {}^{\vee}x{=}5.$$

The postcondition of the second assignment (using the predicate transformer described above) reduces as follows

$$\lambda P[\exists z[\{z/{}^{\vee}p\}{}^{\vee}P \vee {}^{\vee}p{=}x]({}^{\wedge}[{}^{\vee}x{=}5]) = \exists z[\{z/{}^{\vee}p\}({}^{\vee}x{=}5) \wedge$$
$$\wedge\ {}^{\vee}p{=}x] = [{}^{\vee}x{=}5 \wedge {}^{\vee}p{=}x].$$

Finally, the postcondition of the last assignment reduces as follows:

$$\lambda P\exists z[\{z/{}^{\vee}x\}{}^{\vee}P \wedge {}^{\vee}x{=}6]({}^{\wedge}[{}^{\vee}x{=}5 \wedge {}^{\vee}p{=}x]) = \exists z[\{z/{}^{\vee}x\}({}^{\vee}x{=}5 \wedge {}^{\vee}p{=}x) \wedge {}^{\vee}x{=}6] =$$
$$[\exists z[z{=}5 \wedge {}^{\vee}p{=}x] \wedge {}^{\vee}x{=}6] = [{}^{\vee}p{=}x \wedge {}^{\vee}x{=}6].$$

From this formule follows ${}^{\vee\vee}p{=}6$, so the postcondition has as consequence that the integer value related with p is 6. This is as it should be (see figure 2).

If we compare the treatment of this program with the treatment using Floyd's rule see (18)-(23), then we observe that this success is due to a careful distinction between the representation of the interpretation of identifier x, namely x, and the representation of the *value* of that identifier, namely ${}^{\vee}x$. This has as its effect that in the calculation of the last postcondition the x in the identity $p{=}x$ is not replaced by z as would be the case if Floyd's rule were used.
6.1. END

For the constants which translate pointers, we have postulates analogous to the ones we have for constants translating integer identifiers (rigidness postulate, distinctness postulate, update postulate). Something more, however, has to be said about the possible values of pointer constants. Consider $p \in CON_{<s,<s,e>>}$. This constant is interpreted as a function from states to objects of type <s,e>. Not all such objects are possible values of pointers. In a given state the extension of p has to be the interpretation of some integer identifier and we have already formulated some requirements concerning such interpretations (update postulate etc.). For instance, the interpretation of an integer identifier cannot be a constant function yielding for all states the same value. Consequently the extension of p cannot be such an object. Thus we arrive at the following postulate concerning the constants of type <s,<s,e>> (for higher order pointers analogous requirements will be given).

6.2. <u>Properness postulate</u>

For all $c \in CON_{<s,<s,e>>}$, $s \in ST$

$$V(c)(s) \in \{V(c') \mid c' \in CON_{<s,e>}\}.$$

6.2. END

6.2. <u>Arrays</u>

In section 3 it was shown that a straightforward application of Floyd's rule to assignments containing subscripted array identifiers may yield incorrect results. Here a compositional treatment of the semantics of such assignments will be developed (the formal treatment will be given in 7). In order to have a comparision for the treatment, I will first sketch a treatment due to De BAKKER (1976, 1980).

De BAKKER presents an extension of Floyd's proof rule for the case of assignment statements. His treatment is based on the definition of a new kind of substitution operator $[\alpha/\beta]$. In most cases this operator behaves as the ordinary substitution operator, but not in the case that both α and β are of the form array-identifier-with-subscript. Then this substitution may result in a compound expression containing an *if-then-else* construction. The relevant clause of the definition of the operator is as follows.

(41) $$\begin{cases} [t/a[s_1]](b[s_2]) = b[[t/a[s_1]](s_2)] \\ [t/a[s_1]](a[s_2]) = \underline{if}\ [t/a[s_1]]s_2 = s_1\ \underline{then}\ t\ \underline{else}\ a[[t/a[s_1]]s_2]. \end{cases}$$

Using this operator, De Bakker gives a variant of Floyd's rule for assignment statements:

(42) $$\{\phi\}\ a[s] := t\ \{\exists y \exists z[[y/a[z]](\phi) \wedge z = [y/a[z]](s) \wedge a[z] = [y/a[z]](t)]\}.$$

6.3. <u>EXAMPLE</u>.
Assignment: $a[a[1]] := 2$. Precondition: $a[1] = 1 \wedge a[2] = 1$.
Postcondition:
$\exists y \exists z[[y/a[z]](a[1] = 1 \wedge a[2] = 1) \wedge z = [y/a[z]](a[1]) \wedge a[z] = [y/a[z]](2)]$.
By the definition of substitution this reduces to
$\exists y \exists z[\underline{if}\ 1{=}z\ \underline{then}\ y\ \underline{else}\ a[1]\ \underline{fi} = 1 \wedge \underline{if}\ 2{=}z\ \underline{then}\ y\ \underline{else}\ a[2]\ \underline{fi} = 1 \wedge$
$z = \underline{if}\ 1{=}z\ \underline{then}\ y\ \underline{else}\ a[1]\ \underline{fi} \wedge a[z]{=}2]$.
From the second and the third boolean expression in the conjunction, we see that we must take $z{=}1$, and the whole expression reduces to:
$\exists y[y{=}1 \wedge a[2]{=}1 \wedge 1{=}y \wedge a[1]{=}2]$.
This is in turn equivalent to $a[1]{=}2 \wedge a[2]{=}1$, from which it follows that $a[a[1]]{=}2$.
This proof rule works correctly! It is not easy to understand why the rule works, but De Bakker has proven its correctness.
6.3. END

From our methodological point of view this solution has the same disadvantages as Floyd's original proposal, the main one being that the substitution operator defined in (41) has no semantic interpretation. In order to obtain a solution within the limits of our framework, let us consider the 'parts' of the assignment $a[s] := t$. The usual syntax says that there are two parts: the left hand side, (i.e. $a[s]$), and the right hand side (i.e. t). The left hand side has as its parts an array identifier (i.e. a) and an integer expression (i.e. s). This analysis has as a consequence that, in our algebraic approach, we have to associate with the array identifier a some semantic object. In the papers by De Bakker this is not done, nor is this done by several other authors in the field. One usually employs a model which is an abstract computer model with cells, and it is not possible to associate some cell with a. In our model, on the other hand, it is not difficult to associate some semantic object with a. For each state an array identifier determines a function from integers (subscripts) to integers (the value contained in the cell with that subscript). In analogy to the treatment of integer identifiers, this relation between an identifier and the associated function, is given by translating the array identifier into a rigid constant of type <s,<e,e,>>.

Using the fact that it makes sense to speak about the value of (the translation of) an array identifier a, we can easily describe the effect of the execution of an assignment $a[\beta] := \gamma$. By this assignment a state is reached in which the value associated with a differs for one argument from its old value. If the old value of a is denoted by z, then the new value of a is, roughly, described by: $\lambda n[\underline{if}\ n{=}\beta\ \underline{then}\ \gamma\ \underline{else}\ z(n)\ \underline{fi}]$. I said 'roughly' since it is not yet expressed, for β and γ, to take here the old value of a. These considerations give rise to the following predicate transformer associated with $a[\beta] := \gamma$:

(43) $\lambda P[\exists z\{z/{}^{\vee}a\}{}^{\vee}P \wedge {}^{\vee}a = \{z/{}^{\vee}a\}(\lambda n[\underline{if}\ n = \beta'\ \underline{then}\ \gamma\ \underline{else}\ {}^{\vee}a[n]\ \underline{fi}])]$.

Notice the direct analogy of this predicate transformer with the predicate transformer for the simple assignment. The correctness of (43) is, I believe, much clearer than of the one given by De Bakker. This perspicuity is due to the fact that we treat the array identifiers as having a meaning. In a model based upon the use of 'cells', such an approach does not come naturally. The main point of the present approach (arrays as functions) is the basis for the treatment of arrays in GRIES 1977. It turned out that already in HOARE & WIRTH 1973 arrays are considered as denoting functions (however not in the context of the problems under discussion).

6.4. EXAMPLE. Consider the assignment $a[a[1]] := 2$, executed in a state in which $a[1] = 1$ and $a[2] = 1$. We wish to find the strongest postcondition in this situation. This is found by application of the predicate transformer (associated with the assignment) to the precondition expressing the mentioned property of the state. In the logical formulas given below I should write $a(1)$ etc., since we interpret a as a function. But in order to keep in mind what we are modelling, I prefer the notation $a[1]$

$$[a[a[1]] := 2]'({}^{\wedge}[[{}^{\vee}a[1]=1 \wedge {}^{\vee}a[2]=1]) =$$
$$= \lambda P[\exists z\{z/{}^{\vee}a\}{}^{\vee}P \wedge {}^{\vee}a = \{z/{}^{\vee}a\}(\lambda n\ \underline{if}\ n={}^{\vee}a[1]\ \underline{then}\ 2\ \underline{else}\ {}^{\vee}a[n]\underline{fi})]({}^{\wedge}[{}^{\vee}a[1]=1 \wedge {}^{\vee}a[2]=1])$$
$$= \exists z[z[1]=1 \wedge z[2]=1 \wedge {}^{\vee}a = \lambda n\ \underline{if}\ n=z[1]\ \underline{then}\ 2\ \underline{else}\ z[n]\underline{fi}]$$
$$= \exists z[z[1]=1 \wedge z[2]=1 \wedge {}^{\vee}a = \lambda n\ \underline{if}\ n=1\ \underline{then}\ 2\ \underline{else}\ z[n]\underline{fi}].$$

From this postcondition the value of $a[a[1]]$ can be calculated:

$$^{\vee}[a[^{\vee}a[1]] = {}^{\vee}a[\lambda n\ \underline{if}\ n{=}1\ \underline{then}\ 2\ \underline{else}\ z[n]\ \underline{fi}\ [1]] = {}^{\vee}a[2] = z[2] = 1.$$

6.4. END

Now that we know which predicate transformer should be used, let us look at how it was obtained. We could have tried to find some translation for the left hand side of the assignment (i.e. for $a[n]$), out of which the predicate transformer could be formed. It turned out to be preferable to use the insights obtained from considerations based on the principle of compositionality. We observed that $a[s] := t$ is a notation for changing the function associated with a. This suggests to consider such an assignment as a three-place syntactic operation which takes as inputs the array identifier, the subscript expression, and the expression at the right hand side of the := sign. In such an approach it is easy to obtain the desired predicate transformer, and therefore this approach will be followed. This shows that semantic considerations may influence the design of the syntactic rules (however, a binary approach to the assignment is not forbidden).

In JANSSEN & Van EMDE BOAS (1977a) assignments to multi-dimensional arrays are treated. Since the proposal given there, is not strictly in accordance with the principle of compositionality, it is not mentioned here. One could incorporate assignments to n-dimensional arrays by introducing a separate rule for each choice of n; then an n-dimensional array is considered as a function of n arguments.

7. SECOND FRAGMENT

7.1. The rules

In this section I will present the syntax and semantics of a fragment of the programming language ALGOL-68 (Van WIJNGAARDEN 1975). The fragment contains integer identifiers, pointers to integer identifiers, pointers to such pointers, etc., so there is in principle an infinite hierarchy of pointers. The fragment also contains arrays of integers, arrays of integer identifiers, arrays of pointers to integer identifiers, etc., so in principle an infinite hierarchy of arrays. In order to deal with such infinite sets, the syntax will contain rule schemata. These schemata are like the hyperrules used in the official ALGOL-68 report (VAN WIJNGAARDEN 1975). Following the framework from ch.1, the semantics of the fragment will be described by means of a translation into intensional logic. As explained in the previous section, this logic has to be interpreted in a restricted class of models. The models have to satisfy certain postulates; these will be presented in section 7.3. In section 7.4 a model will be constructed that satisfies these postulates.

The names of the categories used in section 5 have to be changed in order to follow the ALGOL-68 terminology. The category of integers will be called 'int id' (i.e. integer identifier) and the category of integer identifiers ID will be called 'ref int id' (i.e. reference to integer identifier). As explained above there will be an infinite set of categories. In the description of a category name we may use the meta notion <u>mode</u>. The possible substitutions for this metanotion are described by the following meta rules;

<u>mode</u> → int
<u>mode</u> → ref <u>mode</u>
<u>mode</u> → row of <u>mode</u>.

These modes correspond with types of intensional logic; this correspondence is formalized by the mapping τ which is defined as follows.

$$\begin{aligned}
&\tau(\text{int}) &&= e\\
&\tau(\text{bool}) &&= t\\
&\tau(\text{ref }\underline{\text{mode}}) &&= \langle s,\tau(\underline{\text{mode}})\rangle\\
&\tau(\text{row of }\underline{\text{mode}}) &&= \langle e,\tau(\underline{\text{mode}})\rangle.
\end{aligned}$$

For each 'mode' there is a category 'mode id' which contains denumerable many expressions: the identifiers of that mode. Examples are:

<u>Category</u>	<u>Typical identifiers</u>
int id	$1,2,3,\ldots,666,\ldots$
ref int id	$x,y,w,x_1,x_2,\ldots$
ref ref int id	$p,q,p_1,p_2,\ldots$
row of int id	$a,a_1,a_2,\ldots$

The rule schemata of the fragment are presented in the same way as the rules presented in section 5. The main difference is that in section 5 we had actual rules, whereas we here have schemata which become actual rules by means of a substitution for <u>mode</u>. Important is that throughout one scheme the same substitution for <u>mode</u> has to be used.

Rule $B_1..B_5$	: int exp $\times$ int exp $\rightarrow$ bool exp
$FB_1..FB_5$	: $\alpha \divideontimes \beta$ where $\divideontimes$ stands for $<,>,\leq,\geq$, or $=$.
$TB_1..TB_5$	: $\alpha' \divideontimes \beta'$ idem for $\divideontimes$.
Rule $I_1..I_3$	: int exp $\times$ int exp $\rightarrow$ int exp
$FI_1..FI_3$	: $\alpha \oplus \beta$ where $\oplus$ stands for $+,\times,\div$ respectively
$TI_1..TI_1$	: $\alpha' \oplus \beta'$ idem for $\oplus$.
Rule E_1	: $\underline{\text{mode}}$ id $\rightarrow$ $\underline{\text{mode}}$ unit
FE_1	: α
TE_1	: α'.
Rule E_2	: $\underline{\text{mode}}$ unit $\rightarrow$ $\underline{\text{mode}}$ exp
FE_2	: α
TE_2	: α'.
Rule E_3	: ref $\underline{\text{mode}}$ exp $\rightarrow$ $\underline{\text{mode}}$ exp
FE_3	: α
TE_3	: ${}^{\vee}\alpha'$.
Rule E_4	: bool exp $\times$ $\underline{\text{mode}}$ unit $\times$ $\underline{\text{mode}}$ unit $\rightarrow$ $\underline{\text{mode}}$ unit
FE_4	: $\underline{\mathit{if}}\ \alpha\ \underline{\mathit{then}}\ \beta\ \underline{\mathit{else}}\ \gamma\ \underline{\mathit{fi}}$
TE_4	: $\underline{\mathit{if}}\ \alpha'\ \underline{\mathit{then}}\ \beta'\ \underline{\mathit{else}}\ \gamma'\ \underline{\mathit{fi}}$
comment	: The rule is defined for units and not for exp's in order to avoid the problems of 'balancing' (see e.g. Van WIJNGAARDEN 1975).
Rule E_5	: ref row of $\underline{\text{mode}}$ unit $\times$ int exp $\rightarrow$ ref $\underline{\text{mode}}$ unit
FE_5	: $\alpha'[\beta']$
TE_5	: ${}^{\wedge}[{}^{\vee}\alpha'[\beta']]$.
Rule A_1	: ref $\underline{\text{mode}}$ id $\times$ $\underline{\text{mode}}$ exp $\rightarrow$ ass
FA_1	: $\alpha := \beta$
TA_1	: $\lambda P \exists z[\{z/{}^{\vee}\alpha'\}{}^{\wedge}P \wedge {}^{\vee}\alpha' = \{z/{}^{\vee}\alpha'\}\beta']$ where $z \in VAR_{\tau(\underline{\text{mode}})}$.
Rule A_2	: ref row of $\underline{\text{mode}}$ id $\times$ int exp $\times$ $\underline{\text{mode}}$ exp $\rightarrow$ ass
TA_2	: $\alpha[\beta] := \gamma$
FA_2	: $\lambda P[\exists z\{z/{}^{\vee}\alpha'\}{}^{\vee}P \wedge {}^{\vee}\alpha' = \{z/{}^{\vee}\alpha'\}[\lambda n\ \underline{\mathit{if}}\ n{=}\beta'\ \underline{\mathit{then}}\ \gamma'\ \underline{\mathit{else}}\ z[n]\ \underline{\mathit{fi}}]]$.

Rule P_1 : ass $\to$ simple prog
FP_1 : α
TP_1 : α'.

Rule P_2 : simple prog $\to$ prog
FP_2 : α
TP_2 : α'.

Rule P_3 : prog $\times$ simple prog $\to$ prog
FP_3 : $\alpha;\beta$
TP_3 : $\lambda P(\alpha'({}^{\wedge}[\beta'(P)]))$.

Rule P_4 : bool exp $\times$ prog $\times$ prog $\to$ prog
FP_4 : $\underline{if}\ \alpha\ \underline{then}\ \beta\ \underline{else}\ \gamma\ \underline{fi}$
TP_4 : $\lambda P[\beta'({}^{\wedge}[\alpha' \wedge {}^{\vee}P]) \vee \gamma'({}^{\wedge}[\neg\alpha' \wedge {}^{\vee}P])]$.

7.1. <u>EXAMPLE</u>. In section 5 I have given several examples of assignment statements. Therefore now as example a somewhat more complex program

$p := a[1];\ a[1] := 2$ precondition $a[1]=1 \wedge a[2]=2$.

The postcondition after the first assignment is:

$$[p := a[1]]'({}^{\wedge}[{}^{\vee}a[1]=1 \wedge {}^{\vee}a[2]=2]) =$$
$$\lambda P\exists z[\{z/{}^{\vee}p\}{}^{\vee}P \wedge {}^{\vee}p = \{z/{}^{\vee}p\}{}^{\wedge}[{}^{\vee}a[1]]]({}^{\wedge}[{}^{\vee}a[1]=1 \wedge {}^{\vee}a[2]=2]) =$$
$$[{}^{\vee}a[1]=1 \wedge {}^{\vee}a[2]=2 \wedge {}^{\vee}p = {}^{\wedge}[{}^{\vee}a[1]]].$$

Then the postcondition after the second assignment is:

$$\exists z\{z/{}^{\vee}a\}({}^{\vee\wedge}[{}^{\vee}a[1]=1 \wedge {}^{\vee}a[2]=2 \wedge {}^{\vee}p = {}^{\wedge}[{}^{\vee}a[1]]]) \wedge$$
$${}^{\vee}a = \{z/{}^{\vee}a\}[\lambda n\ \underline{if}\ n = 1\ \underline{then}\ 2\ \underline{else}\ a[n]\ \underline{fi}] \quad =$$
$$= \exists z[z[1]=1 \wedge z[2]=2 \wedge {}^{\vee}p = {}^{\wedge}[{}^{\vee}a[1]] \wedge$$
$${}^{\vee}a = \lambda n\ \underline{if}\ n = 1\ \underline{then}\ 2\ \underline{else}\ z[n]\ \underline{fi}].$$

From this we conclude that ${}^{\vee\vee}p = {}^{\vee}a[1] = 2$.
7.1. END

7.2. <u>The postulates</u>

In order to formulate the postulates, I will first define the set AT of achievable types. This set consists of the types which are achievable by translating expressions of categories which have a name obtained from the name-scheme '<u>mode</u> exp'.

7.2. <u>DEFINITION</u>. The set $AT \subset Ty$ is defined by the following clauses:

1. $e \in AT$
2. If $\tau \in AT$ then $<s,\tau> \in AT$ and $<e,\tau> \in AT$.

7.2. END

The rigidness postulate says that all constants are rigid designators.

7.3. <u>Rigidness Postulate</u>

For all $\tau \in AT$ and $c \in CON_{<s,\tau>}$: $\exists v \Box [c=v]$.

7.3. END

The distinctness postulate says that two states are different only if they give rise to a different extension of some constant.

7.4. <u>Distinctness Postulate</u>

Let $s,t \in ST$. If for all $\tau \in AT$ and $c \in CON_{<s,\tau>}$ we have $V(c)(s) =$ $= V(c)(t)$, then $s = t$.

7.4. END

The properness postulate says, roughly, that the extension of a constant has to be a value that can be achieved by executing instructions from the programming language. First we define these sets AV_τ of achievable values of type τ.

7.5. <u>DEFINITION</u>. The sets $AV_\tau (\tau \in AT)$ of *achievable values* of type τ are defined as the smallest sets satisfying the following clauses.

I $AV_e = \mathbb{N}$

II $\{V(c) \mid c \in CON_{<s,\tau>}\} \subset AV_{<s,\tau>}$

III if $\rho \in AV_{<s,<e,\tau>>}$ and $n \in \mathbb{N}$ then $\lambda s[[\rho(s)](n)] \in AV_{<s,\tau>}$

IV $AV_{<e,\tau>} = AV_\tau^{\mathbb{N}}$.

7.6. <u>Properness Postulate</u>

For all $s \in ST, \tau \in AT$, $c \in CON_{<s,\tau>}$ we have $V(c)(s) \in AV_\tau$.

7.6. END

The update postulate says that the model should have such a richness that the value of one identifier can be changed into arbitrary achievable value, without changing the values of other identifiers.

7.8. Update Postulate

For all $s \in ST, \tau \in AT$, $c \in CON_{<s,\tau>}$, $d \in AV_\tau$, there is a state $t \in ST$ such that

1. $V(c)(t) = d$
2. $V(c')(t) = V(c')(s)$ for all constants $c' \not\equiv c$.

7.8. END

The update postulate only requires 'updating' to an achievable value. This means that the interpretation of $\{\alpha/{}^{\vee}c\}$ can be defined as follows.

7.9. DEFINITION.

$$V_{s,g}\{\alpha/{}^{\vee}c\}\phi = \begin{cases} V_{<c \leftarrow V_{s,g}\alpha>s,g}\ \phi & \text{if } V_{s,g}(\alpha) \text{ is achievable} \\ V_{s,g}\phi & \text{otherwise.} \end{cases}$$

7.9. END

7.3. A model

The postulates concerning the model can be distinguished in two groups. Some of the postulates require a certain richness of the model (the distinctness postulate and the update postulate), other postulates limit this richness (rigidness postulate and properness postulate). I will show that it is possible to steer a course between this Scylla and Charibdis by constructing a model which satisfies all these postulates.

The model will be built from the set of natural numbers and a set of states. This set of states should have a certain richness since the model has to fulfill the update postulate (every constant can take every achievable value). In order to obtain this effect one would like to take as set of states the cartesian product of the sets AV_τ of achievable values of type τ. This method cannot be used since the achievable values themselves are defined using the set of states (clause III of their definition). Therefore we will first introduce a collection of expressions which will turn out to be in a one-one correspondence with the achievable values. The set of states will be defined as the cartesian product of the sets of these expressions.

7.11. DEFINITION. The sets $AE_\tau (\tau \in AT)$ of achievable value expressions of type τ are defined as the smallest sets satisfying the following clauses:

(1) $\forall\ i \in CON_e$ if $\underline{i} \in AE_e$

(2) $\forall\ c \in CON_{<s,\tau>}$ if $c \in AE_{<s,\tau>}$

(3) $\forall\ i \in AE_e$ and for $\forall\ \rho \in AE_{<s,<e,\tau>>}$: $\underline{\rho[i]} \in AE_{<s,\tau>}$

(4) If for all $n \in \mathbb{N}$: $\phi_n \in AE_\tau$ then $(\phi_n)_{n\in\mathbb{N}} \in AE_{<e,\tau>}$.

7.10. END

Clause (4) introduced infinite sequences of symbols. They arise since we did not formalize the finiteness of arrays. The above definition has as a consequence that corresponding to each achievable value given by the definition of AV, there is an expression in AE.

A model for IL satisfying the postulates is now constructed as follows. We use the sets AE_τ of achievable value denotations and define the set of states by

$$S = \prod_{\tau\in AT} \prod_{CON_{<s,\tau>}} AE_\tau .$$

For $c \in CON_{<s,\tau>}$ we denote the projection on the c-th coordinate of a state s by $\Pi_c(s)$.

Having chosen the set S, the sets D_τ are determined for each type τ. To complete the description of the model we must explain how $V(c)$ is defined for constants. This function is defined simultaneously with a mapping $G: \bigcup_{\tau\in AT} AE_\tau \rightarrow \bigcup_{\tau\in AT} AV_\tau$.

(1) $V(i) = G(\underline{i}) = i$ for $\underline{i} \in AE_e$

i.e. a number denotations are mapped onto the integers denoted by them.

(2) $V(c) = G(\underline{c}) = \underline{\lambda} s[G(\Pi_c(s))]$ for $c \in CON_{<s,\tau>}$

(3) $G(\underline{\rho[i]}) = \underline{\lambda} s[G(\rho)(s)[G(\underline{i})]]$ for $\rho \in AE_{<s,<e,\tau>>}$

(4) $G((\phi_n)_{n\in\mathbb{N}}) = \underline{\lambda} n[G(\phi_n)]$ for $(\phi_n)_{n\in\mathbb{N}} \in AE_{<e,\tau>}$.

Clearly the map $G: \bigcup_{\tau\in AT} AE_\tau \rightarrow \bigcup_{\tau\in AT} AV_\tau$ in this way becomes a bijection. So all elements in the model which are of an achievable type, are achievable values. Moreover the model satisfies all postulates, due to the definition of the set S.

8. CORRECTNESS AND COMPLETENESS

8.1. State transition semantics

In the previous section the meaning a program is defined, and one might expect that the story ends there. But the kind of meanings (predicate transformers) are far removed from the behaviour of a computer while executing a program. One might ask whether we did not loose the connection with a notion of meaning that is more connected with the behaviour of computers. In order to answer this question another kind of semantics will be considered; one in which the meanings of assignments and programs are defined as mappings from states to states, rather than as predicate transformers. I will call it a state-transition semantics; it is related with the standard denotational semantics.

In order to express such a state transition semantics, we need a language in which states can be represented. In the present context the best choice seems to be Ty2: two sorted type theory (see chapter 3, or GALLIN 1975, for a definition). For our purposes this language is extended with state switchers:

8.1. DEFINITION. If $\tau \in AT$, $c \in CON_{<s,\tau>}$, $\beta \in ME_\tau$ and $s \in ME_s$, then $<c \leftarrow \beta>s \in ME_s$. The interpretation of such an expression is defined by

$V_g(<c\leftarrow\beta>s) = V_{g'}(s)$, where $g' \underset{s}{\sim} g$ and $g'(s)$ is the unique state t, such that $V(c)(t) = V(\beta)$ and $V(c')(t) = V(c')(s)$ if $c' \not\equiv c$, if such a state exists (the update postulate guarantees uniqueness); otherwise $g'(s) = s$.

8.1. END

The state-transition semantics of the fragment is defined by means of providing for a translation into Ty2. The translation function will be denoted as ". For the identifiers the translation into Ty2 is the same as the translation into IL, so $\chi' = \chi''$ for all identifiers χ.

For most of the translation rules into Ty2 the formulation can easily be obtained from the translation rules into IL using the standard formulation of IL in Ty2 (see chapter 3). Therefore I will present here only those rules which are essentially different: the rules concerning assignments and programs.

Rule P_1 : Ass $\rightarrow$ Simple Prog
FP_1 : α
OP_1 : α''.

Rule P_2 : Simple Prog $\rightarrow$ Prog
FP_2 : α
OP_2 : α''.

Rule P_3 : Prog $\times$ Simple Prog $\rightarrow$ Prog
FP_3 : $\alpha ; \beta$
OP_3 : $\lambda s[\alpha''(\beta''(s))]$.

Rule P_4 : Bool Exp $\times$ Prog $\times$ Prog $\rightarrow$ Prog
EP_4 : $\underline{\mathit{if}}\ \alpha\ \underline{\mathit{then}}\ \beta\ \underline{\mathit{else}}\ \gamma\ \underline{\mathit{fi}}$
OP_4 : $\lambda s[\underline{\mathit{if}}\ \alpha''(s)\ \underline{\mathit{then}}\ \beta''\ \underline{\mathit{else}}\ \gamma''\ \underline{\mathit{fi}}]$.

Rule A_1 : Ref $\underline{\text{mode}}$ Id $\times$ $\underline{\text{mode}}$ Exp $\rightarrow$ Ass
FA_1 : $\alpha := \beta$
OA_1 : $\lambda s[<\alpha'' \leftarrow \beta''>(s)]$.

Rule A_2 : Ref Row of $\underline{\text{mode}}$ Id $\times$ Int Exp $\times$ $\underline{\text{mode}}$ Exp $\rightarrow$ Ass
TA_2 : $\alpha[\beta] := \gamma$
OA_2 : $\lambda s[<\alpha'' \leftarrow \lambda n\ \underline{\mathit{if}}\ n = \beta''\ \underline{\mathit{then}}\ \gamma''\ \underline{\mathit{else}}\ \alpha''[n]\ \underline{\mathit{fi}}>(s)]$.

8.2. Strongest postconditions

Our aim is to prove that the predicate transformers we have defined in the previous section, are correct with respect to the operational semantics", and that these predicate transformers give as much information about the final state as possible. The relevant notions are defined as follows.

8.2. DEFINITION. A forward predicate transformer π' is called *correct* with respect to program π if for all state predicates ϕ and all states s.

if $s \models \phi$ then $\pi''(s) \models \pi'(^\wedge\phi)$.

8.3. DEFINITION. A forward predicate transformer π' is called *maximal* with respect to program π if for all pairs of state predicates ϕ,ψ holds:

if for all states s: $s \models \phi$ implies $\pi''(s) \models \psi$,
then $\models \pi'(^\wedge\phi) \rightarrow \psi$.

8.4. <u>THEOREM</u>. *Let π be a program, and π_1 and π_2 be forward predicate transformers which are correct and maximal with respect to π. Then for all ϕ:*

$$\models \pi_1(^\wedge\phi) \leftrightarrow \pi_2(^\wedge\phi).$$

<u>PROOF</u>. Since π_2 is correct we have:

if $s \models \phi$ then $\pi''(s) \models \pi_2(^\wedge\phi)$.

Since π_1 is maximal, from the above implication follows:

$$\models \pi_1(^\wedge\phi) \rightarrow \pi_2(^\wedge\phi).$$

Analogously we prove

$$\models \pi_2(^\wedge\phi) \rightarrow \pi_1(^\wedge\phi).$$

8.4. END

A consequence of this theorem is that all predicate transformers which are correct and maximal with respect to a certain program yield equivalent postconditions. This justifies the following definition.

8.5. <u>DEFINITION</u>. Let π be a program and ϕ an expression of type t. Now $sp(\pi,\phi)$ is a new expression of type t, called the *strongest postcondition* with respect to π and ϕ. The interpretation of $sp(\pi,\phi)$ is equal to the interpretation of $\pi'(^\wedge\phi)$, where π' is a forward predicate transformer which is correct and maximal with respect to π.

8.5. END

A notion which turns out to be useful for proving properties of predicate transformers is

8.6. <u>DEFINITION</u>. A predicate transformer π' is called *recoverable* with respect to program π if for all states t and state-predicates ϕ

if $t \models \pi'(^\wedge\phi)$ then there is a state s such that $s \models \phi$ and $\pi''(s) = t$.

8.7. <u>THEOREM</u>. *If π' is recoverable, then π' is maximal.*

<u>PROOF</u>. Suppose that π' is recoverable and assume that $s \models \phi$ implies that $\pi''(s) \models \psi$, but that not $\models \pi'(^\wedge\phi) \rightarrow \psi$. Then there is a state t such that $t \models \pi(^\wedge\phi)$ and $t \models \neg\psi$. Since π' is recoverable there is a state s such that $s \models \phi$ and $\pi''(s) = t$. By assumption we also have $\pi''(s) \models \psi$. Contradiction.

8.8. <u>THEOREM</u>. *The translation function ' defined in section 7 yields strongest postconditions.*

<u>PROOF</u>. By induction to the structure of the possible programs. We only consider the case $\chi := \delta$ because for other cases the proof is straightforward.

<u>Part 1: Correctness</u>. Let $s \models \phi$ and $t = \pi''(s)$. Thus $t = \langle\chi \leftarrow \delta''\rangle s$. We have to prove that

(44) $t \models \exists z[\{z/{}^{\vee}\chi'\}\phi \wedge {}^{\vee}\chi' = \{z/{}^{\vee}\chi'\}\delta']$.

Let h be such that $h(z) = V_s({}^{\vee}\chi)$. Then for every formula ψ:

(45) $V_{t,h}(\{z/{}^{\vee}\chi'\}\psi) = V_{\langle\chi'\leftarrow h(z)\rangle t,h}(\psi) = V_{s,h}(\psi)$.

Therefore

(46) $t,h \models \{z/{}^{\vee}\chi'\}\phi$.

Moreover

(47) $V_{t,h}(\{z/{}^{\vee}\chi'\}\delta') = V_{\langle\chi\leftarrow h(z)\rangle t,h}\ \delta' = V_{s,h}\ \delta' = V_{t,h}\ \chi'$.

This means that ' is correct.

<u>Part 2: Recoverability</u>. Let

(48) $t \models \exists z[\{z/{}^{\vee}\chi\}\phi \wedge {}^{\vee}\chi = \{z/{}^{\vee}\chi'\}\delta']$.

Thus there is a g such that (49) and (50) hold

(49) $t,g \models \{z/{}^{\vee}\chi'\}\phi$

(50) $V_t({}^{\vee}\chi') = V_{t,g}(\{z/{}^{\vee}\chi'\}\delta')$.

We define $s = \langle\chi\leftarrow g(z)\rangle t$, then we immediately conclude that $s \models \phi$. We prove now that the value of ${}^{\vee}\chi'$ is the same in $\pi''(s)$ and in t. Since this is the only identifier in which they might differ we conclude that the states are the same (the update postulate guarantees uniqueness!)

(51) $V_{\pi''(s)}\ ({}^{\vee}\chi') = V_{\langle\chi\leftarrow\delta''\rangle s}\ ({}^{\vee}\chi') = V_s(\delta') = V_{\langle\chi\leftarrow g(z)\rangle t}(\delta') =$
$= V_{t,g}\{z/{}^{\vee}\chi'\}\delta' = V_t({}^{\vee}\chi')$.

Notice that this proof also holds in case that δ is an λ-expression, or in case g(z) is not achievable. This means that π' is recoverable, hence π' is maximal.

8.8. END

8.3. Completeness

The notions 'completeness' and 'soundness' of a collection proof rules play an important role in the literature concerning the semantics of programming languages. Such collections are intended to be used for proving properties of programs. Our main aim was not to prove properties, but to define meanings. However, in the discussions about our approach the possibility to prove properties of programs played an important role. In the examples several proofs concerning effects of programs were given, and one of the arguments for using predicate transformers was their usefulness in proofs. Therefore it is interesting to consider our approach in the light of the notions 'soundness' and 'completeness'. First I will informally discuss these notions in their relation to the traditional approach (for a survey see APT 1981), there after I will try to transfer them to our approach.

In the traditional approaches one describes the relation between the assertions (state predicates) ϕ and ψ and the program π by means of the correctness formula $\{\phi\}\pi\{\psi\}$. This formula should be read as stating that if ϕ holds before the execution of program π, then ψ holds afterwards (for a discussion see section 2). Formula ϕ is called a *precondition*, and ψ a *postcondition*. collection C of proof rules for such formulas consists of axioms, and of proof rules which allow to derive new formulas from already derived ones. For the basic constructions of the programming language certain formulas are given as axioms (e.g. Floyd's axiom for the assignment statement). An important proof rule is (52); the so called *rule of consequence*. It allows us to replace a precondition by a stronger statement, and a postcondition by a weaker statement.

(52) If $p \rightarrow p_1$, $\{p_1\}S\{q_1\}$, $q_1 \rightarrow q$ are derived, then $\{p\}S\{q\}$ follows.

The notion $\vdash_C$ (derivable in C) is then defined as usual. Hence (53) means that the formula $\{\phi\}\pi\{\psi\}$ can be derived from the axioms by using only rules from C.

(53) $\vdash_C \{\phi\}\pi\{\psi\}$.

Besides the syntactic notion $\vdash_C$, the semantic notion $\models_M$ of satisfaction in a model M is used. A model M is defined, in which assertions ϕ and ψ can be interpreted and in which the execution of π is modelled. Then (54) says that it is true in M that if ϕ holds before the execution of π, then

ψ holds afterwards.

(54) $\models_M \{\phi\}\pi\{\psi\}$.

The notions soundness and completeness of collection C of proof rules relate the syntactic notion $\vdash_C$ with the semantic notion $\models_M$. The collection C is called *sound* if for all ϕ,ψ and π

(55) $\vdash_C \{\phi\}\pi\{\psi\}$ implies $\models_M \{\phi\}\pi\{\psi\}$.

The collection C is called *complete* if for all ϕ,ψ and π

(56) $\models_M \{\phi\}\pi\{\psi\}$ implies $\vdash_C \{\phi\}\pi\{\psi\}$.

Most identifiers in computer programs have to be associated with numbers, and the assertions in correctness formulas may say something about the numerical values of these identifiers. We may consider a trivial program α (e.g. $x := x$) a trivial assertion β (e.g. *true*), and an arbitrary assertion γ from number theory. Then (57) holds.

(57) $\models_M \{\beta\}\alpha\{\gamma\}$ if and only if $\models_M \gamma$.

Suppose now that we had a complete collection C of proof rules for correctness formulas. Then combination of (56) with (57) would learn us that (58) holds

(58) $\vdash_C \{\beta\}\alpha\{\gamma\}$ if and only if $\models_M \gamma$.

Thus a complete proof system for correctness formulas would give us a complete proof system for arithmetic. Since arithmetic is not completely axiomatizable, there cannot be such a complete system C for correctness formulas. Concerning this situation De BAKKER (1980, p.61) says the following:

'[..] *we want to concentrate on the programming aspects of our language, and* [..] *pay little attention to questions about assertions which do not interact with* [*assignment*] *statements (so that even if an axiomatization of validity were to exist, we might not be interested in using it)*.

For this reason De Bakker takes all valid assertions as axioms of C, i.e. if $\models_M \phi$, then by definition $\vdash_C \phi$. This notion of completeness, viz. where certain assertions are taken as axioms, is called *complete in the sense of Cook*. For a formal definition see COOK (1978), or APT (1981). This notion is defined only for logical languages which are *expressive*: languages in which all strongest postconditions can be expressed (for the class of programs under consideration). From the results in 8.2 follows that our extension of IL is expressive.

In order to define the notions 'soundness' and 'completeness' for our approach, we have to find notions that can be compared with $\vdash_C$ and with $\models_M$. First I will consider the syntactic notion $\vdash_C$. In our approach the logical deductions are performed on the level of intensional logic. So if we would introduce a system S of proof rules, it would be proof rules of intensional logic. Hence we have to find an expression of IL which corresponds with (52).

We have characterized (in intensional logic) the meaning of a program π by means of a predicate transformer π', and we have proven that this transformer yields strongest postconditions, i.e. $sp(\pi,\phi) = \pi'(^\wedge\phi)$. Consider now (59)

(59) $\pi'(^\wedge\phi) \rightarrow \psi$.

Formula (59) expresses that if ϕ holds before the execution of π, then ψ holds afterwards. So (59) corresponds with the correctness formula $\{\phi\}\pi\{\psi\}$. An alternative approach would of course be to use the corresponding backward predicate transformer. The discussion below will be restricted to forward predicate transformers; for backward predicate transformers related remarks could be made. Suppose now that we have a system S of proof rules of intensional logic. The notion $\vdash_S$ can be defined as usual. Then (60) says about S the same as (53) says about C. Therefore I will consider (60) as the counterpart of (53).

(60) $\vdash_S \pi'(^\wedge\phi) \rightarrow \psi$.

In section 7 we have defined a class of models. Let $\models$ denote the interpretation in these models. In the light of the above discussion (61) can be considered as the counterpart in our system of (54).

(61) $\models \pi'(^\wedge\phi) \rightarrow \psi$.

A system of proof rules for IL is called *sound* if for all ϕ,ψ and π (62) holds.

(62) $\vdash_S \pi'(^\wedge\phi) \rightarrow \psi$ implies $\models \pi'(^\wedge\phi) \rightarrow \psi$.

A system S of proof rules is called *complete* if for all ϕ,ψ and π (63) holds

(63) $\models \pi'(^\wedge\phi) \rightarrow \psi$ implies $\vdash_S \pi'(^\wedge\phi) \rightarrow \psi$.

We might consider again trivial program α, trivial condition β, and an arbitrary IL formula δ. Then (64) holds

(64) $\models \alpha'(^\wedge\beta) \rightarrow \delta$ if and only if $\models \delta$.

Suppose now that proof system S contains modus ponens. Then (65) holds

(65) $\vdash_S \alpha'(^\wedge\beta) \rightarrow \delta$ if and only if $\vdash_S \delta$.

Suppose moreover that S is complete. Then from (64) and (65) it follows that (66) holds

(66) $\models \delta$ if and only if $\vdash_S \delta$.

Thus a complete system of proof rules would give us a complete axiomatization of IL. Such an axiomatization does not exist (see chapter 3). Hence S cannot be complete either. In this situation we might follow De Bakker, and make the notion of completeness independent of the incompleteness of the logic we use. So we might take all formulas of our extension of IL as axioms. But then S is complete (in the sense of Cook) in a trivial way since all correctness formulas are formulas of our extension of IL.

This completeness result is not very exciting, and one might try to find another notion of completeness. A restriction of the axioms to only arithmetical assertions seems me to be unnatural for the fragment under discussion because our programs do not only deal with natural numbers, but also with pointers of different kinds. From a logical viewpoint it is attractive to try to prove for our extension a kind of generalized completeness (see chapter 3). This would require that Gallin's axiom system for IL (see chapter 3) is extended with rules concerning state-switchers. Thus we might show that a system S is generalized complete, i.e. that it is complete with respect to the formulas which are true in all generalized models. The models defined in section 7 constitute a subclass of the set of generalized models. I do not know any reason to expect that the formulas valid in all models of this subclass are the same as those valid in all generalized models (because our subclass does not contain an important class: the standard models). Hence generalized completeness would be an interesting result that proves a certain degree of completeness, but it would not correspond with the traditional completeness results in computer science. I doubt whether computer scientists would be happy with such a completeness result.

Another concept between 'incomplete' and trivially 'complete', is suggested by Peter van Emde Boas. The formula $\pi'(^\wedge\phi) \rightarrow \psi$ was intended to be the analogue of the Hoare formula $\{\phi\}\pi\{\psi\}$. The language in which we express ϕ and ψ contains state switchers, but in most cases a programmer will be interested in cases were ϕ and ψ are state-switcher free. However, our analogue of the Hoare formula, viz. $\pi'(^\wedge\phi) \rightarrow \psi$, will always contain a state-switcher introduced by the predicate transformer π'. Now one might hope for a result which says that this state-switcher can always be eliminated. In the examples we described this was indeed the case. There are however situations where no reduction rule is applicable (if values of pointers are involved, where these values are unknown). This makes it unlikely that it will always be possible to eliminate the state-switcher from a formula obtained by application of a predicate transformer to a state-switcher free formula (i.e. such an expressibility result is not te be expected). It would however, be interesting to know whether the reduction formulas are sufficient to eliminate the state-switchers from those translations of Hoare formulas where elimination is possible. This gives the following intermediate concept of 'completeness'.

If ϕ and ψ are state-switcher free and $\models \pi'(^\wedge\phi) = \psi$ then $\vdash_S \pi'(^\wedge\phi) = \psi$.

9. THE BACKWARD APPROACH

9.1. Problems with Hoare's rule

Besides the approach discussed up till now, there is the approach based on backward predicate transformers. In section 3 we have already met Hoare's rule for the assignment statement

(67) $\{[\delta/v]\ \psi\}\ v := \delta\ \{\psi\}$.

Hoare's rule may yield incorrect results when applied to assignment containing pointers or arrays, just as was the case with Floyd's rule. I mention three examples.

De BAKKER (1976) presents for Hoare's rule the following example

(68) $\{[1/a[a[2]]](a[a[2]]=1)\}\ a[a[2]] := 1\ \{a[a[2]]=1\}$.

The precondition in (68) reduces to $1=1$. That would imply that, for any initial state, the execution of $a[a[2]] := 1$ has the effect that afterwards $a[a[2]] = 1$ holds. This is incorrect (consider e.g. an initial state satisfying the equality $a[2]=2 \wedge a[1]=2$).

GRIESS (1977) presents the following example

(69) $\{1=a[j]\}\ a[i] := 1\ \{a[i] = a[j]\}$.

Whereas in example (68) the obtained precondition was too weak, in the present example the obtained precondition is too restrictive. The postcondition holds also in case the initial state satisfies $i=j$.

An example of the failure of Hoare's rule for the treatment of pointers is (JANSSEN & VAN EMDE BOAS 1977b):

(70) $\{x=x+1\}\ p := x;\ \{p=x+1\}\ x := x+1\ \{p=x\}$.

It is impossible to satisfy the precondition mentioned in (70), whereas for any initial state the postcondition will be satisfied.

Besides the objection that (67) gives incorrect results in certain cases, the same more fundamental problems arise as were mentioned in section 3 for Floyd's rule (e.g. the use of textual substitution).

9.2. Backward predicate transformers

Using a state switcher a formulation can be given for the backward predicate transformers which satisfies our algebraic framework. The transformer corresponding to $v := \delta$ is

$$\lambda P[\{\delta'/{}^{\vee}v\}{}^{\vee}P].$$

The transformer corresponding with $a[\beta] := \gamma$ is

$$\lambda P[\{\lambda n\ \underline{if}\ n = \beta'\ \underline{then}\ \gamma'\ \underline{else}\ {}^{\vee}a[n]\ \underline{fi}/{}^{\vee}a\}{}^{\vee}P].$$

9.1. EXAMPLE. Assignment $a[i] := 1$; Postcondition ${}^{\vee}a[i] = {}^{\vee}a[j]$
Precondition: $\{\lambda n\ \underline{\text{if}}\ n = i\ \underline{\text{then}}\ 1\ \underline{\text{else}}\ {}^{\vee}a[n]\underline{\text{fi}}/{}^{\vee}a\}({}^{\vee}a[i]={}^{\vee}a[j])$ reducing to:
$1 = (\underline{\text{if}}\ j = i\ \underline{\text{then}}\ 1\ \underline{\text{else}}\ a[j]\underline{\text{fi}})[j]$ and
further to: $j = i \vee a[j] = 1$ (compare this with (69)).

9.2. <u>EXAMPLE</u>. Assignment $a[a[2]] := 1$; postcondition ${}^{\vee}a[{}^{\vee}a[2]] = 1$.
Precondition: $\{\lambda n\ \underline{if}\ n = {}^{\vee}a[2]\ \underline{then}\ 1\ \underline{else}\ a[n]\ \underline{fi}/{}^{\vee}a\}({}^{\vee}a[{}^{\vee}a[2]]=1)$.
We have to apply the state switcher to both occurrences of ${}^{\vee}a$ in the postcondition. If we apply it to ${}^{\vee}a[2]$ then we obtain

$$\underline{if}\ 2 = {}^{\vee}a[2]\ \underline{then}\ 1\ \underline{else}\ {}^{\vee}a[2]\ \underline{fi}.$$

This leads us to consider the following two cases.

I. $2 = {}^{\vee}a[2]$.
Then the precondition reduces to
$(\{\lambda n\ \underline{if}\ n = a[2]\ \underline{then}\ 1\ \underline{else}\ a[n]\ \underline{fi}/{}^{\vee}a\}{}^{\vee}a)[1] = 1$ which reduces to
$a[1] = 1$.

II. $2 \neq {}^{\vee}a[2]$.
Then the precondition reduces to
$(\{\lambda n\ \underline{if}\ n = a[2]\ \underline{then}\ 1\ \underline{else}\ a[n]\ \underline{fi}/{}^{\vee}a\}{}^{\vee}a)[2] = 1$ which reduces to
$1 = 1$.

So the precondition is $({}^{\vee}a[2] = 2 \wedge {}^{\vee}a[1] = 1) \vee ({}^{\vee}a[2] \neq 2)$. (Compare this result with (68)).
9.2. END

9.3. Weakest preconditions

We aim at obtaining backward predicate transformers which yield a result that is correct with respect to the operational semantics ", and which require assumptions as weak as possible about the initial state (i.e. dual to the requirements concerning the forward predicate transformers). The relevant notions are defined as follows.

9.3. DEFINITION. A backward predicate transformer $'\pi$ is called *correct* with respect to a program π if for all state predicates ϕ and all states s

$$\text{if } s \models {}'\pi({}^{\wedge}\phi) \quad \text{then} \quad \pi''(s) \models \phi.$$

9.4. DEFINITION. A backward predicate transformer $'\pi$ is called *minimal* with respect to a program π if for all pairs of state predicates η and ϕ, the following holds:

$$\text{if for all states } s\text{: } s \models \eta \text{ implies } \pi''(s) \models \phi,$$
$$\text{then } \models \eta \rightarrow {}'\pi({}^{\wedge}\phi).$$

9.5. THEOREM. *Let π be a program, and π_1 and π_2 be backward predicate transformers which are correct and minimal with respect to π. Then for all ϕ:*

$$\models \pi_1({}^{\wedge}\phi) \leftrightarrow \pi_2({}^{\wedge}\phi).$$

PROOF. Since π_1 is correct, we have:

$$\text{if } s \models \pi_1({}^{\wedge}\phi) \quad \text{then} \quad \pi''(s) \models \phi.$$

Since π_2 is minimal, from this implication follows

$$\models \pi_1({}^{\wedge}\phi) \rightarrow \pi_2({}^{\wedge}\phi).$$

Analogously we prove $\models \pi_2({}^{\wedge}\phi) \rightarrow \pi_1({}^{\wedge}\phi)$.
9.5. END

A consequence of this theorem is that all backward predicate transformers which are correct and minimal with respect to a certain program, yield equivalent preconditions. This justifies the following definition.

9.6. DEFINITION. Let π be a program and ϕ a state predicate. Then $wp(\pi,\phi)$ is a new expression of type t, called the weakest precondition with respect to π and ϕ. The interpretation of $wp(\pi,\phi)$ is equal to the interpretation of $'\pi(^{\wedge}\phi)$, where $'\pi$ is a backward predicate transformer which is correct and minimal with respect to π. If a state predicate is equivalent with $wp(\pi,\phi)$ it is called a weakest precondition with respect to π and ϕ.
9.6. END

In 9.2 backward predicate transformers are defined for the assignment statements. We wish to prove that they yield a weakest precondition. This will not be proven in a direct way because it turns out that backward and forward predicate transformers are closely related. The one can be defined from the other, and correctness and maximality of the forward predicate transformers implicate correctness and minimality of the backward predicate transformers. These results will be proven in the next subsections.

9.4. Strongest and weakest

Strongest postconditions and weakest preconditions can syntactically be defined in terms of each other. This connection is proved in the following theorem.

9.7. THEOREM. *Let* $Q \in \mathrm{VAR}_{<s,t>}$ *and let*

(I) *be* $\exists Q[^{\vee}Q \wedge \Box[sp(\pi,{}^{\vee}Q) \rightarrow \phi]]$

and

(II) *be* $\forall Q[\Box[\phi \rightarrow wp(\pi,{}^{\vee}Q)] \rightarrow {}^{\vee}Q]$.

Then it holds that
formula (I) *is equivalent to* $wp(\pi,\phi)$, *and formula* (II) *is equivalent to* $sp(\pi,\phi)$.

PROOF.
part A
I show that (I) is correct (A_1) and minimal (A_2) with respect to ϕ,π and ".
From this follows that (I) is equivalent to $wp(\pi,\phi)$.

part A_1
Suppose that s satisfies (I), so

$$s \models \exists Q[^{\vee}Q \wedge \Box[sp(\pi,{}^{\vee}Q) \rightarrow \phi]].$$

Then for some g (71) and (72) holds

(71) $s,g \models {}^{\vee}Q$

(72) $s,g \models \Box[sp(\pi,{}^{\vee}Q) \rightarrow \phi]$.

By definition of *sp*, from (71) follows

$$\pi''(s),g \models sp(\pi,{}^{\vee}Q).$$

By definition of $\Box$ from (72) follows

$$\pi''(s),g \models sp(\pi,{}^{\vee}Q) \rightarrow \phi.$$

Therefore

$\pi''(s),g \models \phi$, or equivalently $\pi''(s) \models \phi$.

This means that (I) is correct.

part A_2
Suppose that for all s holds

(73) $s \models \eta$ implies $\pi''(s) \models \phi$.

By definition of *sp* from (73) follows

$$\models sp(\pi,\eta) \rightarrow \phi.$$

So for all s

$$s \models \Box[sp(\pi,\eta) \rightarrow \phi].$$

Let g be an assignment such that $g,s \models Q = {}^{\wedge}\eta$. Then

$$s,g \models \Box[sp(\pi,{}^{\vee}Q) \rightarrow \phi].$$

So (for the choice $Q = {}^{\wedge}\eta$)

$$s \models \eta \rightarrow \exists Q[{}^{\vee}Q \wedge \Box[sp(\pi,{}^{\vee}Q) \rightarrow \phi]].$$

This means that (II) is minimal.

part B
I show that (II) is correct (B_1) and maximal (B_2) with respect to ϕ,π, and".
From this it follows that (II) is equivalent with $sp(\pi,\phi)$.

part B_1
Let

$$s \models \phi.$$

Suppose that for variable assignment g holds

(74) $g \models \Box[\phi \rightarrow wp(\pi,{}^{\vee}Q)]$.

Then from (74) follows

$s,g \models wp(\pi,{}^{\vee}Q)$.

So

(75) $\pi''(s),g \models {}^{\vee}Q$.

From (74) and (75) it follows that for all g holds

$\pi''(s),g \models \Box[\phi \to wp(\pi,{}^{\vee}Q)] \to {}^{\vee}Q$.

So

$\pi''(s) \models \forall Q[\Box[\phi \to wp(\pi,{}^{\vee}Q] \to {}^{\vee}Q]$.

This means that (II) is correct.

part B_2
Suppose that for all s holds

(76) $s \models \phi$ implies $\pi''(s) \models \eta$.

Then, by definition of *wp* it follows that

(77) $\models \phi \to wp(\pi,\eta)$.

Suppose that t satisfies (II), so

(78) $t \models \forall Q[\Box[\phi \to wp(\pi,{}^{\vee}Q)] \to {}^{\vee}Q]$.

Let g be an assignment such that $g,t \models Q = {}^{\wedge}\eta$. Then:

(79) $t,g \models \Box[\phi \to wp(\pi,{}^{\vee}Q)] \to {}^{\vee}Q$.

Then from (77) and (79) follows

(80) $t,g \models {}^{\vee}Q$.

So

(81) $\models \forall Q[[\Box[\phi \to wp(\pi,{}^{\vee}Q)] \to {}^{\vee}Q] \to {}^{\vee}Q]$.

This means that II is maximal.
9.7. END

That $wp(\phi,\pi)$ and $sp(\pi,\phi)$ are closely connected is also observed by RAULEFS (1978). He gives a semantic connection. Theorem 9.7 goes further, because an explicit syntactic relation is given.

9.5. Correctness proof

The theorem 9.7 has as a consequence that weakest preconditions and strongest postconditions can be defined in terms of each other. Now it is unlikely that the formulas with quantification over intensions of predicates are the kind of expressions one would like to handle in practice. The importance of the theorem is that given some expression equivalent with $sp(\pi,\phi)$, it allows us to prove that some expression (found on intuitive considerations) is equivalent with $wp(\pi,\phi)$. From the correctness and maximality of the predicate transformers defined in section 5 and 7, it follows that the backward predicate transformers defined in this section are correct and minimal.

9.8. THEOREM. *The following two statements are equivalent*

(I) $sp(\chi := \delta,\ \phi) = \exists z[\{z/{}^{\vee}\chi\}\phi \wedge {}^{\vee}\chi' = \{z/{}^{\vee}\chi'\}\delta']$

(II) $wp(\chi := \delta,\ \phi) = \{\delta'/{}^{\vee}\chi\}\phi$.

PROOF.

part 1: (I) $\Rightarrow$ (II).

Assume that (I) holds. Then from theorem 9.7 follows:

(82) $\models wp(\chi := \delta,\phi) = \exists Q[{}^{\vee}Q \wedge \Box[\exists z[\{z/{}^{\vee}\chi'\}{}^{\vee}Q \wedge {}^{\vee}\chi' = \{z/{}^{\vee}\chi'\}\delta'] \rightarrow \phi]]$.

So we have to prove that for arbitrary assertion ϕ and state s holds that

(83) $s \models \{\delta'/{}^{\vee}\chi'\}\phi$

if and only if

(84) $s \models \exists Q[{}^{\vee}Q \wedge \Box\ \exists z[\{z/{}^{\vee}\chi'\}{}^{\vee}Q \wedge {}^{\vee}\chi' = \{z/{}^{\vee}\chi'\}\delta'] \rightarrow \phi]]$

part 1a: (83) $\Rightarrow$ (84)

Assume that (83) holds. Let g be a variable assignment such that

(85) $g \models Q = {}^{\wedge}\{\delta'/{}^{\vee}\chi'\}\psi$.

Then (due to (83)) we have

(86) $s,g \models {}^{\vee}Q$.

In order to prove (84) we have next to prove the necessary validity of the formula mentioned after the $\Box$ for this choice of Q. So we have to prove that for arbitrary state t (87) implies (88).

(87) $t \models \exists z[\{z/{}^{\vee}\chi'\}\{\delta'/{}^{\vee}\chi'\}\phi \wedge {}^{\vee}\chi' = \{z/{}^{\vee}\chi'\}\delta']$

(88) $t \models \phi$.

Let h be a variable assignment for which (87) holds. Then using the iteration theorem, we find

(89) $t,h \models \{\{z/{}^{\vee}\chi'\}\delta'/{}^{\vee}\chi'\}\phi \wedge {}^{\vee}\chi' = \{z/{}^{\vee}\chi'\}\delta'$.

The second conjunct gives us information about the value of ${}^{\vee}\chi'$ in this state. The state switcher says that we have to interpret ϕ with respect to the state where ${}^{\vee}\chi'$ precisely has that value. So the state switcher does not change the state! This means that

(90) $t \models \phi$.

So (87) implies (88), and therefore (84) holds.

part 1b: (84) $\Rightarrow$ (83)
Assume (84) holds. Then there is a variable assignment g such that (91) and (92) hold

(91) $s,g \models {}^{\vee}Q$

(92) $s,g \models \Box[\exists z[\{z/{}^{\vee}\chi'\}{}^{\vee}Q \wedge {}^{\vee}\chi' = \{z/{}^{\vee}\chi'\}\delta'] \rightarrow \phi]$.

In (92) it is said that a certain formula is necessarily valid. Application of this to state $<\chi'\leftarrow\delta'>s$ gives

(93) $<\chi'\leftarrow\delta'>s,g \models \exists z[\{z/{}^{\vee}\chi'\}{}^{\vee}Q \wedge {}^{\vee}\chi' = \{z/{}^{\vee}\chi'\}\delta'] \rightarrow \phi$.

Let $g' \underset{z}{\sim} g$ be such that $g'(z) = V_s({}^{\vee}\chi')$ so $<\chi'\leftarrow z><\chi'\leftarrow\delta'>s = s$. Since (91) holds, we have

(94) $<\chi'\leftarrow z><\chi'\leftarrow\delta'>s,g' \models {}^{\vee}Q$.

Consequently

(95) $<\chi'\leftarrow\delta'>s,g' \models \{z/{}^{\vee}\chi'\}{}^{\vee}Q$.

Moreover

(96) $<\chi'\leftarrow\delta'>s,g' \models {}^{\vee}\chi' = \{z/{}^{\vee}\chi'\}\delta'$

because $V_{<\chi'\leftarrow\delta'>s}({}^{\vee}\chi') = V_s(\delta') = V_{<\chi'\leftarrow z><\chi'\leftarrow\delta'>s}(\delta') = V_{<\chi'\leftarrow\delta'>s}\{z/{}^{\vee}\chi'\}\delta'$.
From (94) and (95) follows that the antecedent of the implication in (93) holds. Therefore the consequent of the implication holds

(97) $<\chi'\leftarrow\delta'>s,g' \models \phi$

so

(98) $s,g' \models \{\delta'/\chi'\}\phi$.

This means that (83) holds, so (84) ⇒ (83). And this completes the proof of (I) ⇒ (II).

part 2: (II) ⇒ (I)

The proof of (II) ⇒ (I) uses a related kind of arguments. Therefore this proof will be presented in a more concise way. Assume that (II) holds. Then we have to prove that for arbitrary s and ϕ:

(99) $s \models \forall Q[\Box[\phi \rightarrow \{\delta'/{}^{\vee}\chi'\}{}^{\vee}Q] \rightarrow {}^{\vee}Q]$

if and only if

(100) $s \models \exists z[\{z/{}^{\vee}\chi'\}\phi \wedge {}^{\vee}\chi' = \{z/{}^{\vee}\chi'\}\delta']$.

part 2a

Assume (99). Take for Q in (99) the assertion in (100). We now prove that the antecedent of (99) holds, then (100) is an immediate consequent. So suppose $t \models \phi$. We have to prove that

(101) $t \models \exists z[\{\delta'/{}^{\vee}\chi'\}\{z/{}^{\vee}\chi'\}\phi \wedge \{\delta'/{}^{\vee}\chi'\}[{}^{\vee}\chi' = \{z/{}^{\vee}\chi'\}\delta']]$

or equivalently

(102) $t \models \exists z[\{z'/{}^{\vee}\chi'\}\phi \wedge \delta' = \{z/{}^{\vee}\chi'\}\delta']$.

This is true for $g(z) = V_t({}^{\vee}\chi')$, so the antecedent of (99) holds, and from this follows that (100) holds.

part 2b

Assume (100). Let g be arbitrary and assume

(103) $s,g \models \phi \rightarrow \{\delta'/{}^{\vee}\chi'\}{}^{\vee}Q$.

This is the antecedent of (99). We now prove that the consequent holds, so that

(104) $s,g \models {}^{\vee}Q$.

From (100) follows that for some $g' \underset{z}{\sim} g$

(105) $<\chi' \leftarrow z>s,g' \models \phi$.

Using (103), from (105) follows

(106) $<\chi' \leftarrow z>s,g' \models \{\delta'/{}^{\vee}\chi'\}{}^{\vee}Q$.

Consequently

(107) $s,g' \models \{\{z/{}^{\vee}\chi'\}\delta'/{}^{\vee}\chi'\}{}^{\vee}Q$.

From (100) also follows

(108) $s,g' \models {}^{\vee}\chi' = \{z/{}^{\vee}\chi'\}\delta'$.

From (108 and (107) we may conclude

(109) $s,g' \models {}^{\vee}Q$.

This proves (104), so (99) follows from (100).
9.8. END

9.9. THEOREM. *The following two statements are equivalent*

(I) $\models sp(\alpha[\beta] := \delta,\phi) = \exists z[\{z/{}^{\vee}\alpha'\}\phi \wedge {}^{\vee}\alpha' = \{z/{}^{\vee}\alpha'\}\lambda n\ \underline{if}\ n = \beta'\ \underline{then}\ \delta'\ \underline{else}\ {}^{\vee}\alpha\ [n]\ \underline{fi}]$

(II) $\models wp(\alpha[\beta] := \delta,\phi) = \{\lambda n\ \underline{if}\ n = \beta'\ \underline{then}\ \delta'\ \underline{else}\ {}^{\vee}\alpha'[n]\ \underline{fi}/{}^{\vee}\alpha\}\phi$.

PROOF. The expressions at the right hand side of the equality signs are a special case of the corresponding expressions in the previous theorem. So theorem 9.9 follows from theorem 9.8.
9.9. END

From theorems 9.9 and 9.10 it follows that the predicate transformations for the assignment as defined in section 9.2, yield weakest preconditions.

10. MUTUAL RELEVANCE

In this section I will mention some aspects of the relevance of the study of semantics of programming languages to the study of semantics of natural languages, and vice versa. Most of the remarks have a speculative character.

The present chapter constitutes a concrete example of the relevance of the theory of semantics of natural languages to the study of programming languages. Montague's framework was developed for natural languages, but it is used here for programming languages. The notions 'opaque' and 'transparant', well known in the field of semantics of natural languages, turn out to be useful for the study of semantics of programming languages, see section 1. And the logic developed for the semantics of natural languages turned out to be useful for programming languages as well.

In the semantics of natural languages the principle of compositionality is not only the basis of the framework, but also, as will be shown later, a valuable heuristic tool. It helped us to understand already existing solutions. It gives rise to suggestions how to deal with certain problems, and it is useful in finding weak points in proposals from the literature. I expect that the principle can play the same role in the semantics of programming languages. The treatment of arrays in this chapter (see section 6) is an example of the influence of the principle. Below I will give some further suggestions concerning possible relevance of the principle.

Consider the treatment of 'labels' and 'goto-statements' by A. de Bruyn (chapter 7 in De BAKKER 1980). The treatment is rather complex, and not much motivation for it is given. I expect, however, that these phenomena are susceptible to the technique explained in chapter 1: if the meaning of some statement seems to depend on certain factors, then incorporate these factors into the notion of meaning. In this way the notion of 'continuation' (used by de Bruyn) might be more easily explained, and thus the proposal more easily understood.

In De BAKKER 1980, the proof rules for certain constructions make use of devices which are, from a compositional point of view, not attractive. These constructions are assignments to subscripted array identifiers, procedures with parameters, and declarations of identifiers at the beginning of blocks. In the proof rules for these constructions mainly syntactic substitution is used. From a compositional point of view it is not surprising that the semantic treatment of these phenomena is not completely satisfactory. For assignments to array elements an alternative was proposed in section 6, and for blocks a suggestion was made in section 4. A compositional approach to the semantics of procedures with parameters would describe the meaning of a procedure-call as being built from the meaning of the procedure and the meaning of its argument. If this argument is a reference parameter (call by variable), then the argument position is opaque. This suggests that the meaning of such a procedure should be a function which takes as argument an intension.

In the semantics of natural language ideas from the semantics of programming languages can be used. The basic expression in a programming language is the assignment statement. For the computer the assignment statement is a command to perform a certain action. I have demonstrated how the semantics of such commands is dealt with by means of predicate transformers. Inspired by this approach, we might do the same for commands in natural language. Some examples (taken from Van EMDE BOAS & JANSSEN 1978) are given below. Consider the imperative

(110) *John, drink tea.*

Its translation as a predicate transformer would become something like

(111) $\lambda P[{}^{\wedge}(\mathcal{B}({}^{\vee}P) \wedge \textit{drink-tea}({}^{\wedge}\textit{john})]$.

This expression describes the change of the state of the world if the command is obeyed. The operator $\mathcal{B}$ is a kind of state-switcher, it indicates the moment of utterance of the command. A similar approach can be used to describe the semantics of actions. One might describe the smenatics of performative sentences like

(112) *We crown Charles emperor*

by means of an predicate transformer.

Often a sequence of sentences is used to perform an action rather than to make a some assertions: sentences can be used to give information to the hearer. Consider the text

(113) *Mary seeks John. John is a unicorn.*

These sentences might be translated into the predicate transformers (114) and (115).

(114) $\lambda P[{}^{\vee}P \wedge \textit{seek}_{*}(\textit{mary},\textit{john})]$

(115) $\lambda P[{}^{\vee}P \wedge \textit{unicorn}_{*}(\textit{john})]$.

Suppose that the information the hearer has in the beginning is denoted by ϕ. Then by the first sentence this information is changed into

(116) $\phi \wedge \textit{seek}_{*}(\textit{mary},\textit{john})$

and by the second sentence into

(117) $\phi \wedge \textit{seek}_{*}(\textit{mary},\textit{john}) \wedge \textit{unicorn}_{*}(\textit{john})$.

From the final expression the hearer may conclude that Mary seeks a unicorn.

Also on a more theoretical level the semantics of programming languages can be useful for the study of semantics of natural languages. In the study of natural languages the need for partial functions often arises. In the semantics one wants to use partially defined predicates in order to deal with sortal incorrectness and presuppositions, and in the syntax one wishes to have rules that are not applicable to every expression of the category for which the rule is defined. In the field of programming languages phenomena arise for which one might wish to use partial functions. In this field techniques are used which make it possible to use nevertheless total functions. The basic idea is to introduce in the semantic domain an extra element. Since this approach is, from an algebraic point of view, very attractive, I would like to use this technique in the field of natural languages as well (see chapter 7).

APPENDIX

SAFE AND POLYNOMIAL

In this appendix the theorem will be presented which was announced in chapter 2, at the end of section 7. The theorem states that in an infinitely generated free algebra all safe operations are polynomially definable (free algebras are algebras which are isomorphic to a term algebra). Remind that $f: A_{s_1} \times \ldots \times A_{s_n} \to A_{s_{n+1}}$ is safe in Σ-algebra $<A,F>$ if for every Σ-algebra $<D,G>$ and every $h \in Epi(<A,F>,<D,G>)$ there is a unique $\hat{f}: D_{s_1} \times \ldots \times D_{s_n} \to D_{s_{n+1}}$ such that $h \in Epi(<A,F \cup \{f\}>,<D,G \cup \{\hat{f}\}>)$. The proof originates from F. Wiedijk (pers. comm.).

THEOREM. *Let* $A = <(A_s)_{s\in S},(F_\gamma)_{\gamma\in\Gamma}>$ *be a free algebra, that has a generating set* $(B_s)_{s\in S}$ *where each* B_s *is infinite. Let* $f: A_{s_1} \times \ldots \times A_{s_n} \to A_{s_{n+1}}$ *be a safe operator. Then* f *is a polynomially definable over* A.

HEURISTICS. First I will give some heuristic considerations, there after the theorem will be proved by proving two Lemmas.

Let us assume for the moment that the theorem holds and let us try to reconstruct from f the polynomial p that defines f. Let $<b_1,\ldots,b_n>$ be a possible argument for f, where $b_1,\ldots,b_n$ are generators of A. There is a term $t \in T_{\Sigma,A}$ such that $t_A = f(b_1,\ldots,b_n)$. Since A is free, this term is unique. Hence t is obtained from the polynomial p we are looking for, by means of substituting, for the respective variables in p constants corresponding to $b_1,\ldots,b_n$. Term t (probably) contains constants for $b_1,\ldots,b_n$, but it is not yet clear for any given occurrence of such a constant in t, whether it occurs in p as parameter, or is due to substitution for a variable In order to decide in these matters, we consider the value of f for generators $<c_1,\ldots,c_n>$ which are different from $<b_1,\ldots,b_n>$ and from the constants in t. Suppose that for term u we have $u_A = f(c_1,\ldots,c_n)$. Then u can also be obtained from p by substituting constants. We already know that all constants in p also occur in t. Since $c_1,\ldots,c_n$ do not occur in t, all their occurrences in u are due to of their substitution for variables. So if we replace in u all occurrences of (constants corresponding with) $c_1,\ldots,c_n$ by variables, we have found the polynomial p we were looking for. This idea is followed in the next lemma. We perform these steps and prove that the polynomial so obtained has the desired properties.

<u>LEMMA 1</u>. *There is an infinite sequence* $(z_k)_{k=1,2,\dots}$ *of disjoint* n-*tuples of generators of* A, *and a polynomial* p *such that for each* z_k, $f(z_k) = p(z_k)$.

<u>PROOF</u>. We define by induction a sequence $(z_k)_{k\in\{0,1,\dots\}}$. Let $B_{0,s_1} = B_{s_1},\dots,B_{0,s_n} = B_{s_n}$, and take $z_0 \in B_{0,s_1} \times\dots\times B_{0,s_n}$ arbitrarily. This n-tuple is used for the first attempt to reconstruct the polynomial p which corresponds with f. Below I will define an infinite sequence of attempts to reconstruct p, and there after it will be proved that from the second attempt on always the same polynomial will be found; this is the polynomial p we were looking for, as will be proven in lemma 2.

Assume that $z_0,\dots,z_k$ and $p_0,\dots,p_k$ are already defined. Then we obtain z_{k+1} and p_{k+1} as follows.

Let $C_{k,s}$ be the set of generators of sort s which corresponds with constants in p_k, and let $\{z_{k,s}\}$ be the set of components of z_k of sort s. Define $B_{k,s} = B_{k,s}/(C_{k,s} \cup \{z_{k,s}\})$, and let $z_{k+1} \in B_{k+1,s_1} \times\dots\times B_{k+1,s_n}$. Since A is generated by $(B_s)_{s\in S}$, $f(z_{k+1})$ can be represented by a term t with parameters from $(B_s)_{s\in S}$, including z_{k+1}. This term t can be expressed as a polynomial expression in z_{k+1}, say $p_{k+1}(z_{k+1})$, where, moreover, no component of z_{k+1} occurs as parameter in p_{k+1}. Since for all k and s the sets $C_{k,s}$ and $\{z_{k,s}\}$ are finite, and $B_{k,s}$ was infinite, it follows that $B_{k+1,s}$ is infinite. Hence this construction can be repeated for all k.

Next it will be proven that the polynomials $p_1,p_2,\dots,$ are identical, thus proving the theorem for the sequence $z_1,z_2,\dots,$ (note that p_0 and z_0 are not included). The basic idea of the proof is that we introduce for each k a homomorphism which maps z_k on z_0, and then apply the assumptions of the theorem.

Consider the mapping $\hat{h}$ defined by

$$\begin{cases} \hat{h}(b) = b & \text{if } b \in (B_s/\{z_{k,s}\}) \text{ for some s} \\ \hat{h}(z_k^{(i)}) = z_0^{(i)}, & \text{where } z_k^{(i)} \text{ is the i-th component of } z_k. \end{cases}$$

Since A is free, the mapping $\hat{h}$ determines uniquely a homomorphism $h\colon \langle A,F\rangle \to \langle[(B_s/\{z_{k,s}\})_{s\in S}],F\rangle$. Moreover, h is an epimorphism since all generators of the 'range'-algebra occur in the range of h. The polynomials p_k were chosen to contain no constants corresponding to components of z_k, therefore $h(p_k(z_k)) = p_k(h(z_k))$ holds for all k.

Since operator f is safe, there is a unique $\hat{f}$ such that

$$h \in \mathrm{Epi}(\langle A, F \cup \{f\}\rangle, \langle [B_s, \{z_{k,s}\})_{s \in S}], F \cup \{\hat{f}\}\rangle).$$

Now the following equalities hold:

I $\quad h(f(z_k)) = \hat{f}(h(z_k)) = \hat{f}(z_0) = \hat{f}(h(z_0)) = h(f(z_0)) = h(p_0(z_0)) = p_0(h(z_0)) = p_0(z_0)$.

II $\quad h(f(z_k)) = h(p_k(z_k)) = p_k(h(z_k)) = p_k(z_0)$.

From I and II follows $p_0(z_0) = p_k(z_0)$. Analogously we can prove that $p_1(z_1) = p_k(z_1)$.

Since A is free, there is a unique term t such that $p_k(z_0) = t = p_0(z_0)$. So if we replace the variables in p_k and p_0 by constants corresponding to the components of z_0, we obtain the same expression. From this, and the fact that no components of z_0 occur as constants in p_0, it follows that the constants in p_k consists of:

a1) all the constants in p_0.

a2) possibly some constants corresponding to components of z_0.

Analogously it follows that the constants in p_k consist of

b1) all the constants in p_1

b2) possibly some constants corresponding to components of z_k.

We have chosen z_1 in such a way that no constant in p_0 corresponds to a component of z_1, and no component of z_0 equals a component of z_1. So if p_k contained constants for components of z_1, this would conflict with a1) and a2). Therefore we have to conclude that the constants in p_k are the same as the constants in p_1, and none of these, moreover, corresponds to components of z_1. So for all $k \geq 1$ we have $p_k \equiv p_1$. Call this polynomial p. Then $f(z_k) = p(z_k)$ for all $k \geq 1$.

<u>LEMMA 2</u>. *Let* p *be the polynomial guaranteed by lemma 1. Then for all* $a \in A_{s_1} \times \ldots \times A_{s_n}$, $f(a) = p(a)$.

<u>PROOF</u>. Let $a = \langle a^{(1)}, \ldots, a^{(n)} \rangle$, and assume that $a^{(i)} = t^{(i)}(b_1^{(i)}, \ldots, b_n^{(i)})$, where $t^{(i)}$ is a polynomial without constants, and the $b_j^{(i)}$'s are generators of A. Assume moreover that $f(a) = t$. Let z_k be the infinite sequence of disjoint n-tuples of generators given by lemma 1. Since there are only finitely many constants in t_1 and finitely many $b_j^{(i)}$'s, there is an m such that the components of z_m are all different from the constants in t and the $b_j^{(i)}$'s.

Define

$$\hat{h} \text{ by } \begin{cases} \hat{h}(b) = b & \text{if } b \in B_s/\{z_{m,s}\} \text{ for some } s \\ \hat{h}(z_m^{(i)}) = a^{(i)} & \text{where } z_m^{(i)} \text{ is the i-th component of } z_m. \end{cases}$$

This mapping $\hat{h}$ defines an epimorphism $h \in \mathrm{Epi}(<A,F>,<[(B_s \backslash \{z_{m,s}\})]_{s\in S},F>)$. Since f is safe, there is a unique operation $\hat{f}$ such that

$$h \in \mathrm{Epi}(<A,F \cup \{f\}>,<[(B_s \backslash \{z_{m,s}\}_s)],F \cup \{\hat{f}\}>).$$

Now the following equalities hold

$$f(a^{(1)},\dots,a^{(n)}) = t_A \underset{1}{=} h(t_A) = hf(a^{(1)},\dots,a^{(n)}) =$$
$$h(f(t^1(\vec{b}^{(1)})),\dots,f(t^n(\vec{b}^{(n)}))) \underset{2}{=} \hat{f}(h(t^{(1)}(\vec{b}^{(1)})),\dots,h(t^{(n)}(\vec{b}^{(n)}))) =$$
$$= \hat{f}(h(z_m^{(1)}),\dots,h(z_m^{(n)})) = h(f(z_m^{(1)},\dots,z_m^{(n)})) =$$
$$= h(p(z_m^{(1)},\dots,z_m^{(n)})) = p(h(z_m^{(1)}),\dots,h(z_m^{(2)})) = p(a^{(1)},\dots,a^{(n)}).$$

Equalities 1 and 2 hold since z_m has no components which occur in t or b.
END LEMMAS.

From lemma 1 and lemma 2 the theorem follows

END THEOREM.

INDEX OF NAMES

Adj 17,18,30,37,41-43,66-69,80,83,90,92-94,129,145
Andreka 17
Andrews 111
Apt 172,173
de Bakker 135,140,141,147,158,159,173,175,186
Bartsch 9
van Benthem 33,42,64,80,83,96,100
Birkhoff 43
Blok 33
de Bruyn 186
Burstall 83
Carnap 4
Chany 32
Church 71,100,112
Cook 173,175
Cresswell 3,5,122
Davidson 2
Dijkstra 135,137
Dik 36
Dowty 17
Dummett 5-8,11
Ehrig 83
van Emde Boas 17,37,129,139,140,144,150,161,175,186
Faltz 28
Floyd 135,137,138
Fodor 3,15
Frege 1-11,38
Friedman 120,121
Gabriel 7,8,10
Gallin 100,112,113,122,168
Gazdar 66
Geach 10,38
Gebauer 123
Gerhardt 71
Goguen 83,90,94
Graetzer 33,42,45,46
Gries 140,160
Groenendijk 97,100,115,126
Henkin 32,98,107,112,122
Hermes 7,9,10
van Heyenoort 8
Higgins 43
Hoare 135,136,139,160
Hopcroft 53,55,65
Hughes 122
Huntington 7,10
Jakobovits 143
Janssen 17,36,37,111,125,129,140,150,161,186
Jourdain 10
Karttunen 86
Katz 3,15
Keenan 28
Keisler 33

Kreisel	32
Kreowski	83
Krivine	32
Leibniz	71,108
Lewis	29,34,97
Link	44,100
Lipson	43
Lucas	135
Markusz	17
Marcus	34
Mazurkiewics	4,17
Milner	4,17,37
Monk	32
Montague	3,17,21,27,41,42,44,46,74,76,81,90-92,94,114
Needham	100
Nemeti	129
Neuhold	4,143
Padawitz	83
Partee	3,35,36,97,98
Peremans	79
Peters	17,86
Peano	8
Pietrzykowski	111
Popper	2,5
Potts	144
Pratt	30,131,144
Pullum	66
Quine	71,130
Rabin	52
Rasiova	33
Raulefs	181
Rogers	51
Russell	11
Ryle	143
Sain	17
Schuette	35
Schwartz	15
Scott	4,67
Smith	46
Steinberg	143
Stokhof	97,100,115,126
Stoothoff	10,38
Stoy	4,131,135
Strachey	4,67
Szots	17
Tarski	4,32,35
Tennent	4,131,135,143
Thatcher	56,90
Thiel	8
Thomason	2
Tichy	97
Ullman	53,55,65
Varga	44,100
Veltman	97
Wagner	90
Wall	17
Walk	135

Warren	120,121
Wiedijk	189
Van Wijngaarden	135,161
Wirth	160
Wright	90
Zucker	43,67,147

REFERENCES

Adj, 1977,
=(J.A. Goguen, J.W. Thatcher, E.G. Wagner, J.B. Wright),
'Initial algebra semantics and continuous algebras',
Journal of the Association for Computing Machinery 24, 68-95.

Adj, 1978,
=(J.A. Goguen, J.W. Thatcher, E.G. Wagner),
'An initial algebra approach to the specification, correctness and implementation of abstract data types',
in R. Yeh (ed.), 'Current trends in programming methodology',
Prentice Hall, 1978, pp. 80-149.

Adj, 1979,
=(J.W. Thatcher, E.G. Wagner, J.B. Wright),
'Notes on algebraic fundamentals for theoretical computer science',
in De Bakker & Van Leeuwen 1979, pp. 83-163.

Andreka, H., & I. Nemeti, 1979,
'Applications of universal algebra , model theory and categories in computer science (survey and bibliography),
Computational Linguistics and Computer Languages 13,
251-282. 'Additions' in Computational Linguistics and Computer Languages 14, 7-20.

Andreka, H., & I. Sain, 1981,
'Connections between algebraic logic and initial algebraic semantics of CF languages',
in Domolki & Gergely 1981, pp. 25-83.

Andrews, P.B., 1971,
'Resolution in type theory',
Journal of Symbolic Logic 36, 414-432.

Angelelli, I. (ed.), 1967,
'Gottlob Frege. Kleine Schriften',
Georg Olms, Hildesheim.

Apt, K.R., 1981,
'Ten years of Hoare's logic. Part 1',
A.C.M. Transactions on Programming Languages and Systems 3,
431-483.

Bakker, J.W. de, 1976,
'Correctness proofs for assignment statements',
Report IW 55, Mathematical Centre, Amsterdam.

Bakker, J.W. de & J. van Leeuwen, 1979,
'Foundations of computer science III. part 2: languages, logic, semantics',
Tract 100, Mathematical Centre, Amsterdam.

Bakker, J.W. de, 1980,
'Mathematical theory of program correctness',
Series in Computer Science, Prentice Hall, London.

Bartsch, R., 1978,
'Semantiek en Montague grammatica',
Algemeen Nederlands Tijdschrift voor Wijsbegeerte 70,
117-136.

Benthem, J.F.A.K. van, 1977,
'Tense logic and standard logic',
Logique et Analyse 80, 395-437.

Benthem, J.F.A.K van, 1979a,
'In alle redelijkheid (Openbare les, 29 mei 1979)',
Bulletin centrale interfaculteit Groningen 4,
Universiteit Groningen.
Benthem, J.F.A.K. van, 1979b,
'Universal algebra and model theory. Two excursions on the border',
Report ZW-7908, dept. of math., Groningen University.
Birkhoff, G. & J.D. Lipson, 1970,
'Heterogeneous algebras',
Journal of Combinatorial Theory 8, 115-133.
Blok, W., 1980,
'The lattice of modal logics, an algebraic investigation',
Journal of Symbolic Logic 45, 221-236.
Chang, C.C. & H.J. Keisler, 1973,
'Model theory',
Studies in logic and the foundations of mathematics 73,
North Holland, Amsterdam.
Church, A., 1940,
'A formulation of the simple theory of types',
Journal of Symbolic Logic 5, 56-68.
Church, A., 1956,
'Introduction to mathematical logic. Vol I.',
Princeton Univ. Press, Princeton (N.J.).
Cook, S.A., 1978,
'Soundness and completeness of an axiom system for program verification',
SIAM Journal on Computation 7, 70-90.
Cresswell, M.J., 1973,
'Logics and languages',
Methuen, London.
Davidson, D., 1967,
'Truth and meaning',
Synthese 17, 304-323.
Davidson, D. & G. Harman (eds), 1972,
'Semantics of natural language',
Synthese library 40, Reidel, Dordrecht.
Dijkstra, E.W., 1974,
'A simple axiomatic base for programming language constructs',
Indagationes Mathematicae 36, 1-15.
Dijkstra, E.W., 1976,
'A discipline of programming',
Prentice Hall, Englewood Cliffs (N.J.).
Domolki, B. & T. Gergely (eds), 1981,
'Mathematical logic in programming (Proc. Salgotorjan 1978)',
Colloquia mathematica societatis Janos Bolyai 26,
North Holland, Amsterdam.
Dowty, D.R., R.E. Wall & S. Peters, 1981,
'Introduction to Montague semantics',
Synthese language library 11, Reidel, Dordrecht.
Dummett, M., 1973,
'Frege. Philosophy of language',
Duckworth, London.

Ehrig, H., H.J. Kreowski & P. Padawitz, 1978,
'Stepwise specification and implementation of abstract data types',
in G. Ausiello & C. Boehm (eds), 'Automata, languages and programming (proc. 5 th. coll., Udine)', Lecture notes in computer science 62, Springer, Berlin.

Emde Boas, P. van, 1974,
'Abstract resource-bound classes',
unpublished dissertation, Univ. of Amsterdam.

Emde Boas, P. van & T.M.V. Janssen, 1978,
Montague grammar and programming languages',
in J. Groenendijk & M. Stokhof (eds), 'Proc. of the second Amsterdam coll. on Montague grammar and related topics', Amsterdam papers in formal grammar II, Centrale Interfaculteit, Univ. of Amsterdam, 1978, pp. 101-124.

Emde Boas, P. van, 1978,
'The connection between modal logic and algorithmic logic',
in J. Winkowski (ed.), Mathematical foundations of computer science (7th. symposium Zakopane)', Lecture notes in computer science 64, Springer, Berlin, 1978, pp. 1-18.

Emde Boas, P. van & T.M.V. Janssen, 1979,
'The impact of Frege's principle of compositionality for the semantics of programming and natural languages',
in '"Begriffsschrift". Jenaer Frege-Conferenz 1979',
Friedrich-Schiller Universitaet, Jena, 1979, pp. 110-129.
Also: report 79-07, Dep. of Math, Univ. of Amsterdam, 1979.

Floyd, R.W., 1967,
'Assigning meanings to programs',
in J.T. Schwartz (ed.), 'Mathematical aspects of computer science', Proc. Symp. in Applied Mathematics 19, American Mathematical Society, Providence (R.I.), 1967, pp. 19-32.

Frege, G., 1884,
'Die Grundlagen der Arithmetik. Eine logisch-mathematische Untersuchung ueber den Begriff der Zahl',
W. Koebner, Breslau.
Reprint published by: Georg Olms, Hildesheim, 1961.

Frege, G., 1892,
'Ueber Sinn und Bedeutung',
Zeitschrift fuer Philosophie und philosophische Kritik 100, 25-50.
Reprinted in Angelelli, 1976, pp. 143-162.
Translated as 'On sense and reference' in P.T. Geach & M. Black (eds), 'Translations from the philosophical writings of Gottlob Frege', Basil Blackwell, Oxford, 1952, pp. 56-85.

Frege, G., 1923,
'Logische Untersuchungen. Dritter Teil: Gedankenfuege',
in 'Beitraege zur Philosophie des Deutschen Idealismus. Band III', pp. 36-51.
Reprinted in Angelelli 1976, pp. 378-394.
Translated as 'Compound thoughts' in P.T.Geach & R.H. Stoothoff (transl.), 'Logical investigations. Gottlob Frege', Basil Blackwell, Oxford, 1977, pp. 55-78.

Friedman, J. & D. Warren, 1980,
'Lambda-normal forms in an intensional logic for English'
Studia Logica 39, 311-324.
Gabriel, 1976,
=(G. Gabriel, H. Hermes, F. Kambartel, C. Thiel,
A. Veraart (eds)),
'Gottlob Frege. Wissenschaftliche Briefwechsel',
Felix Meiner, Hamburg.
Gallin, D., 1975,
'Intensional and higher-order modal logic',
Mathematics studies 17, North Holland, Amsterdam.
Gazdar, G., 1982,
'Phrase structure grammar',
in P. Jakobson & G.K. Pullum (eds), 'The nature of syntactic
representation', Synthese language library 15, Reidel, Dordrecht,
1982, pp. 131-186.
Gebauer, H., 1978,
'Montague Grammatik. Eine Einfuerung mit
Anwendungen auf das Deutsche',
Germanistische Arbeitshefte 24, Niemeyer, Tuebingen, 1978.
Gerhardt, C.I. (ed.), 1890,
'Die philosphischen Schriften von Gottfried Wilhelm Leibniz.
Siebenter Band',
Weidmannsche Buchhandlung, Berlin.
Goguen, J.A., & R.M. Burstall, 1978,
Some fundamental properties of algebraic theories:
a tool for semantics of computation.'
Report 53, Department of Artificial Intelligence,
University of Edinburgh.
Graetzer G., 1968,
'Universal algebra',
The univ. series in higher mathematics, van Nostrand, Princeton.
Second edition published by: Springer, New York, 1979.
Gries, D., 1977,
'Assignment to subscripted variables',
Rep. TR 77-305, Dept. of Computer Science, Cornell Univ.,
Ithaca (N,Y.).
Groenendijk J.A.G., T.M.V. Janssen & M.B.J. Stokhof (eds), 1981,
'Formal methods in the study of language. Proceedings of the
third Amsterdam colloquium',
MC-Tracts 135 & 136, Mathematical Centre, Amsterdam, 1981.
Groenendijk, J. & M. Stokhof, 1981,
'Semantics of wh-complements',
in Groenendijk, Janssen, Stokhof, 1981, pp.153 -181.
Groenendijk, J., T.M.V. Janssen, & M. Stokhof (eds), 1984,
'Truth, interpretation, and information. Selected papers from the
third Amsterdam colloquium',
Grass 2, Foris, Dordrecht.
Henkin, L., 1950,
'Completeness in the theory of types',
Journal of Symbolic Logic 15, 81-91.
Henkin, L., 1963,
'A theory of propositional types',
Fundamenta Mathematicae 52, 323-344.

Henkin, L., J.D. Monk & A. Tarski, 1971,
'Cylindric algebras. Part I',
Studies in logic and foundations of mathematics 64,
Norh Holland, Amsterdam.

Hermes, 1969,
=(H. Hermes, F. Kambartel, F. Kaulbach (eds)),
'Gottlob Frege. Nachgelassene Schriften',
Felix Meiner, Hamburg.

Heyenoort, J. van, 1977,
'Sense in Frege',
Journal of Philosophical Logic 6, 93-102.

Higgins, P.J., 1963,
'Algebras with a scheme of operators',
Mathematische Nachrichten 27, 115-132.

Hoare, C.A.R., 1969,
'An axiomatic base for computer programming',
Communications of the Association for Computing Machinery
12, 576-580.

Hoare C.A.R. & N. Wirth, 1973,
'An axiomatic definition of the programming language PASCAL',
Acta Informatica 2, 335-355.

Hopcroft, J.E. & J.D. Ullman, 1979,
'Introduction to automata theory, languages and computation',
Addison-Wesley, Reading (Mass.).

Hughes, G.E. & M.J. Cresswell, 1968,
'An introduction to modal logic',
University Paperbacks 431, Methuen & Co, London.

Janssen, T.M.V. & P. van Emde Boas, 1977a,
'On the proper treatment of referencing, dereferencing
and assignment',
in A. Salomaa & M. Steinby (eds), 'Automata, languages,
and programming (Proc. 4th. coll. Turku)', Lecture notes in
computer science 52, Springer, Berlin, 1977, pp. 282-300.

Janssen, T.M.V. & P. van Emde Boas, 1977b,
'The expressive power of intensional logic in
the semantics of programming languages',
in J. Gruska (ed.), 'Mathematical foundations of
computer science 1977 (Proc. 6th. symp. Tatranska Lomnica)',
Lecture notes in computer science 53, Springer,
Berlin, 1977, pp. 303-311.

Janssen, T.M.V., 1980a,
'Logical investigations on PTQ arising from programming
requirements',
Synthese 44, 361-390.

Janssen, T.M.V., 1981,
'Compositional semantics and relative clause formation
in Montague grammar',
in Groenendijk, Janssen, Stokhof 1981, pp. 237-276.

Janssen, T.M.V., 1981b,
'Montague grammar and functional grammar',
in T. Hoekstra & H v.d. Hulst and M. Moortgat (eds),
'Perspectives on functional grammar', Foris publications,
Dordrecht, 273-297.
also: GLOT 3, 1980, 273-297.

Janssen, T.M.V., & P. van Emde Boas, 1981a,
'On intensionality in programming languages',
in : F. Heny (ed.), 'Ambiguities in intensional contexts',
Synthese library, Reidel, Dordrecht, 1981, pp. 253-269.
Karttunen, L. & Peters, S., 1979,
'Conventional Implicature',
in D.A. Dinneen & C.K. Oh (eds), 'Presuppositions',
Syntax and Semantics 11, Academic Press, New York.
Katz, J.J. & J.A. Fodor, 1963,
'The structure of a semantic theory',
Language 39, pp. 170-210.
Reprinted in J.A. Fodor & J.J. Katz (eds), 'The structure
of language', Prentice Hall, Englewood Cliffs (N.J.), 1964,
pp. 479-518.
Katz, J.J., 1966,
'The philosophy of language',
Harper and Row, London.
Keenan, E.L. & L.M. Faltz, 1978,
'Logical types for natural language',
UCLA occasional papers in linguistics 3.
Kreisel, G. & J.L. Krivine, 1976,
'Elements of mathematical logic. Model Theory',
Studies in logic and the foundations of mathematics 2,
North Holland, Amsterdam.
Kripke, S., 1976,
'Is there a problem about substitutional quantification?',
in G. Evans & J.H. McDowell (eds), 'Truth and meaning.
essays in semantics', Clarendon Press, Oxford, 1976,
pp. 325-419.
Lewis, D., 1970,
'General semantics',
Synthese 22, 18-67.
Reprinted in Davidson & Harman 1972, pp. 169-248.
Reprinted in Partee 1976, pp. 1-50.
Link, G. & M. Varga Y Kibed, 1975,
'Review of R.H. Thomason (ed.), "Formal philosophy. Selected
papers of Richard Montague"',
Erkentniss 9, 252-286.
Lucas, P. & K. Walk, 1971,
'On the formal description of PL/I',
in M.I. Halpern & C.J. Shaw (eds), 'Annual review in
automatic programming, 6', Pergamon Press, Oxford, 1971,
pp. 105-182.
Marcus, R.B., 1962,
'Interpreting quantification',
Inquiry 5, 252-259.
Markusz, Z. & M. Szots, 1981,
'On semantics of programming languages defined by
universal algebraic tools',
in Domolki & Gergely 1981, pp. 491-507.
Mazurkiewicz, A., 1975,
'Parallel recursive program schemes',
in J. Becvar (ed.), 'Mathematical foundations of
computer science (4 th. coll., Marianske Lazne)', Lecture
notes in computer science 32, Springer, Berlin, 1975,
pp. 75-87.

Milner, R., 1975,
'Processes: a mathematical model of computing agents', in H.E. Rose & J.C. Shepherdson (eds), 'Logic colloquium '73 (Bristol)', Studies in logic and the foundations of mathematics 80, North Holland, Amsterdam, pp. 157-173.

Monk, J.D., 1976,
'Mathematical logic',
Graduate texts in mathematics 37, Springer, Berlin.

Montague, R. 1970a,
'English as a formal language',
in Visentini et al., 'Linguaggi nella societa et nella technica', Edizioni di communita, 1970, (distributed by the Olivetti Corporation, Milan).
Reprinted in : Thomason 1974, pp. 188-221.

Montague, R., 1970b,
'Universal grammar',
Theoria 36, 373-398.
Reprinted in : Thomason 1974, pp. 222-246.

Montague, R., 1973,
'The proper treatment of quantification in ordinary English',
in K.J.J. Hintikka, J.M.E. Moravcsik & P. Suppes (eds), 'Approaches to natural language', Synthese Library 49, Reidel, Dordrecht, 1973, pp. 221-242.
Reprinted in Thomason 1974, pp. 247-270.

Needham, P., 1975,
'Temporal perspective. A logical analysis of temporal reference in English',
Dept. of Philosophy, University of Uppsala.

Neuhold, E.J. (ed.), 1978,
'Formal description of programming concepts (IFIP conf. St. Andrews)',
North Holland, Amsterdam.

Partee, B.H., 1973,
'Some transformational extensions of Montague grammar,'
Journal of Philosophical Logic 2, 509-534.
Reprinted in Partee 1976, pp. 51-76.

Partee, B., 1975,
'Montague grammar and transformational grammar',
Linguistic Inquiry 6, 203-300.

Partee, B.H. (ed.), 1976,
'Montague grammar',
Academic Press, New York.

Partee, B.H., 1977b,
'Possible world semantics and linguistic theory',
The Monist 60, 303-326.

Popper, K, 1976,
'Unended quest. An intellectual autobiography',
Fontana.

Pietrzykowski, T., 1973,
'A complete mechanization of second order theory',
Journal of the Association for Computing Machinery 20, pp. 333-365.

Potts, T., 1976,
'Montague's semiotics. A syllabus of errors',
Theoretical Linguistics 3, 191-208.

Pratt, V.R., 1976,
'Semantical considerations on Floyd-Hoare logic',
in 'Proc. 17th. Symp. on Foundations of Computer Science (Houston)', IEEE Computer Society, Long Beach (Cal.), 1976, pp. 109-121.

Pratt, V.R., 1979,
'Dynamic logic',
in De Bakker & Van Leeuwen 1979, pp. 53-82.

Pratt, V.R., 1980,
'Applications of modal logic to programming',
Studia Logica 39, 257-274.

Pullum, G.K. & G. Gazdar, 1982,
'Natural languages and context-free languages',
Linguistics & Philosophy 4, 471-504.

Quine, W.V.O., 1960,
'Word and object',
The MIT Press, Cambridge (Mass.).

Rabin, M., 1960,
'Computable algebra: general theory and the theory of computable fields',
Transactions of the American Mathematical Society, 95, 341-360.

Rasiowa H., 1974,
'An algebraic approach to non-classical logics',
Studies in logic and foundations of mathematics 78,
North Holland, Amsterdam.

Raulefs, P., 1978,
'The connection between axiomatic and denotational semantics of programming languages',
in K. Alber (ed.), 'Programmiersprachen, 5 Fachtagung der G.I.
Univ. Karlsruhe.

Rogers jr., H., 1967,
'Theory of recursive functions and effective computability',
Mc Graw Hill, 1967, New York.

Schuette, K., 1977,
'Proof theory',
Grundlehren der mathematische Wissenschaften 225, Springer, Berlin.

Schwartz, J.T., 1972,
'Semantic definition methods and the evolution of programming languages',
in R. Rustin (ed.), 'Formal semantics of programming languages', Courant Computer Science Symposium 2,
Prentice Hall, Englewood Cliffs (N.J.), 1972, pp. 1-24.

Scott, D. & C. Strachey 1971,
'Towards a mathematical semantics for computer languages',
in J. Fox (ed.), 'Computers and automata (proc. symp. Brooklyn)', Polytechnic Press, Brooklyn (N.Y.), 1971, pp. 16-46.

Steinberg, D.D. & L.A. Jakobovits, 1971,
'Semantics. An interdisciplinary reader in philosophy, linguistics and psychology',
Cambridge Univ. Press.

Stoy, J.E., 1977,
'Denotational semantics: the Scott-Strachey approach to programming language theory',
The MIT Press, Cambridge (Mass.).

Tennent, R.D., 1976,
'The denotational semantics of programming languages',
Communications of the Association for Computing
Machinery 19, 437-453.
Thiel, C., 1965,
'Sinn und Bedeutung in der Logik Gottlob Freges',
Anton Hain, Meisenbach am Glan.
Translated as: 'Sense and reference in Frege's logic',
Reidel, Dordrecht, 1968.
Thomason, R.H. (ed.), 1974,
'Formal philosophy. Selected papers of Richard Montague',
Yale University Press, New Haven.
Tichy, P., 1971,
'An approach to intensional analysis',
Nous 5, 273-297.
Veltman F., 1981,
'Data semantics',
in Groenendijk, Janssen & Stokhof 1981, pp. 541-565.
Revised reprint in Groenendijk, Janssen & Stokhof, 1984,
pp. 43-63.
Wijngaarden, A. van, et al., 1975,
'Revised report on the algorithmic language ALGOL 68',
Acta Informatica 5, 1-236.

CWI Tract 28

Foundations and applications of Montague grammar
Part 2: Applications to natural language

T.M.V. Janssen

Centrum voor Wiskunde en Informatica
Centre for Mathematics and Computer Science

1980 Mathematics Subject Classification: 03B65, 08A40, 08A55, 68S10, 03B45, 03B15.
1982 CR Categories: 1.2.7, F.4.2, J.5.

ISBN 90 6196 306 0

PREFACE

The present volume is one of the two tracts which are based on my dissertation 'Foundations and applications of Montague grammar'. Volume 1 consists of the chapters 1,2,3 and 10 of that dissertation, and volume 2 of the chapters 4-9. Only minor corrections are made in the text. I would like to thank here again everyone who I acknowledged in my dissertation, in particular my promotor P. van Emde Boas, co-promotor R. Bartsch, and coreferent J. van Benthem. For attending me on several (printing-)errors in my dissertation I thank Martin van de Berg, Cor Baayen, Biep Durieux, Joe Goguen, Fred Landman and Michael Moortgat, but in particular Herman Hendriks, who suggested hundreds of corrections. The illustrations are made by Tobias Baanders.

The two volumes present an interdisciplinary study between mathematics, philosophy, computer science, logic and linguistics. No knowledge of specific results in these fields is presupposed, although occasionally terminology or results from them are mentioned. Throughout the text it is assumed that the reader is acquainted with fundamental principles of logic, in particular of model theory, and that he is used to a mathematical kind of argumentation. The contents of the volumes have a lineair structure: first the approach is motivated, next the theory is developed, and finally it is applied. Volume 1 contains an application to programming languages, whereas volume 2 is devoted completely to the consequences of the approach for natural languages.

The volumes deal with many facets of syntax and semantics, discussing rather different kinds of subjects from this interdisciplinary field. They range from abstract universal algebra to linguistic observations, from the history of philosophy to formal language theory, and from idealized computers to human psychology. Hence not all readers might be interested to read everything. Readers only interested in applications to computer science might restrict them selves to volume 1, but then they will miss many arguments in volume 2 which are taken from computer science. Readers only interested in applications to natural language might read chapters 1-3 of volume 1, and all of volume 2, but they will miss several remarks about the connection between the study of the semantics of programming languages and of the semantics of natural languages. Readers familiar with Montague grammar, and mainly interested in practical consequences of the approach, might read chapters 1 and 2 in volume 1 and chapters 6-10 in volume 2, but they will miss new arguments and results concerning many aspects of Montague grammar.

Each chapter starts with an abstract. Units like theorems etc. are numbered (eg 2.3 Theorem). Such a unit ends where the next numbered unit starts, or where the end of the unit is announced (2.3 end). References to collected works are made by naming the first editor. Page numbers given in the text refer to the reprint last mentioned in the list of references, except in case of some of Frege's publications (when the reprint gives the original numbering).

V. The PTQ-fragment 1
1. Introduction 2
2. John runs 4
3. The woman walks 16
4. Mary walks and she talks 20
5. John finds a unicorn 25
6. Every man loves a woman 32
7. Bill walks in the garden 38
8. John tries to find a unicorn 42
9. John believes that Mary will run 47

VI. Variants and deviations 53
1. Introduction 54
2. The use of syntactic information 56
2.1. Introduction 56
2.2. Easy to please 56
2.3. The horse Cannonero 57
3. Non-polynomially defined operators 58
3.1. Introduction 58
3.2. John who runs 59
3.3. Das Mädchen gibt den Apfel dem Vater 61
3.4. Woman such that she loves him 62
4. Operators defined on representants 64
5. New symbols in IL 68
5.1. Introduction 68
5.2. Shake John awake 69
5.3. I and You 70
6. Counting elements 71
6.1. Introduction 71
6.2. Keenan & Faltz count 71
6.3. Partee counts 73
7. The translation language 74
7.1. Introduction 74
7.2. Hausser translates 75
7.3. Lewis translates 76
7.4. Groenendijk & Stokhof translate 76
7.5. Keenan & Faltz on translations 77

VII. Partial rules 79
1. Restrictions of the framework 80
2. Partial algebras 83
2.1. Partial grammars 83
2.2. Partial models 84
2.3. Discussion 87
3. Incorporating transformations 90
4. Defined for another category 93
4.1. Introduction 93
4.2. He$_1$ is loved 94

4.3. Give John a book 94
4.4. Mary shakes John awake again 95
4.5. See himself 96
4.6. Easy to see 98
5. Subcategorization and rule schemes 99
5.1. Hyperrules 99
5.2. Metarules 102
5.3. Variables 103
6. The Well-formedness constraint 108

VIII. Constituent structures 115
1. Structure-why? 116
2. Theoretical aspects 118
2.1. Trees in Montague grammar 118
2.2. Algebraic considerations 121
2.3. Practical differences 122
3. Technical aspects 125
3.1. Introduction 125
3.2. Operations on trees 126
3.3. Features and lexicon 127
3.4. Queries for information 131
4. PTQ syntax 133

IX. Relative clause formation 137
1. Introduction 138
2. The CN-S analysis 138
2.1. The discussion by Partee 138
2.2. The PTQ-rules 140
2.3. Fundamental problems 142
3. The T-S analysis 144
3.1. Cooper on Hittite 144
3.2. Bach & Cooper on English 145
3.3. Fundamental problems 147
4. The proposals of Cooper 148
4.1. Not-there 148
4.2. Left-over, proposal 1 149
4.3. Left-over, proposal 2 150
4.4. Conclusion 154
5. The variable principle 154
6. Many analyses 158
6.1. The CN-S analysis for English 158
6.2. The S-S analysis for Hittite 159
6.3. The T-S analysis for English 161
6.4. The Det-S analysis for English 163
6.5. Conclusion 164
7. Other arguments 165
7.1. Syntax: gender agreement 165
7.2. Semantics: scope 166
7.3. Conclusion 168
8. The general question 169

X. Scope ambiguities of tense, aspect and negation 173
1. Introduction 174
2. The PTQ-approach 174
2.1. Introduction 174
2.2. Syntax of PTQ 175
2.3. Ambiguities 175
2.4. Model 177
3. Basic verb modifiers 177
4. Compound verb modifiers 184
5. Complex constructions 187
5.1. Introduction 187
5.2. Conjoined verb phrases with positive verbs 187
5.3. Conjoined verb phrases with negated verbs 189
5.4. Terms 191
5.5. Embeddings 192
6. One of the rules 194
7. The grammar 199
7.1. Introduction 199
7.2. Rules 200
7.3. Morphology 205
7.4. Fins and Verb Phrase 205
7.5. Final remarks 206

Appendix 1. Individual concepts in PTQ 209
Appendix 2. Set manipulation in syntax 223
Index of names 227
References 229

CHAPTER V

THE PTQ-FRAGMENT

ABSTRACT

In this chapter the fragment of English described in Montague's article PTQ (MONTAGUE 1973) is presented. The method of exposition consists in starting with a very small fragment, and expanding it gradually. In each stage both the syntax and the semantics are discussed extensively. Special attention is paid to the motivation and justification of the analysis.

1. INTRODUCTION

The aim of this chapter is to present in a rigorous way the syntax and the semantics of a certain fragment of a certain dialect of English. The fragment is about the same as the one presented in MONTAGUE (1973), henceforth PTQ. On all essential points I will follow the treatment given in PTQ, in the details, however, there are some differences. The presentation, motivation and justification I will give for the treatment, differs considerably from PTQ. For the presentation I will employ a method which might be called 'concentric'. I will start with a very small fragment, and gradually expand this. For the fragments in each of the stages both the syntax and semantics are given, together with an extensive discussion. I hope that this method will make it easier to understand the sometimes difficult or subtle details of the PTQ-treatment. Certain details (concerning the problems of extension and intension) will be discussed in appendix 1 of this book. A list of the rules of the fragment (useful as a survey) can be found in chapter 8.

In the exposition I will give special attention to algebraic and algorithmic aspects of the treatment. The algebraic considerations often provide an explication why a certain detail is as it is, and not otherwise. The algorithmic aspect concerns the method to obtain simple meaning representations. I do not like some rather abstract relation between a sentence and its meaning. For instance, I am not satisfied with a two-lines-long formula, if there is a one-line-long-formula which represents the same meaning, and if a meaning is represented by a formula which has to be interpreted in models satisfying certain meaning postulates, I would like to have a formula in which these postulates are made explicit. So I prefer concise and clear meaning representations. In order to reach this aim, several rules will be given for the reduction of formulas.

The syntax of the fragment in PTQ is treated rather poorly. In this chapter only minor improvements will be given (for more fundamental changes see chapter 8). But syntax was not Montague's main interest; he was interested primarily in semantics. The fragment is rich in semantically interesting phenomena, and it deals with several famous semantic puzzles. Below I will mention some of the sentences dealt with, together with some comments.

A first kind of phenomena dealt with concerns sentences of which it is clear what their meanings are, and how these should be represented using standard predicate logic. Their challenge lies in the aim to obtain these meanings in a systematic way. Consider (1) and (2).

(1) *John runs.*

(2) *Every man runs.*

These two sentences are closely related in form: only the subject differs. Therefore one would like to produce the sentences along the same lines. The representations of their meanings, however, are rather different. In standard logic it would be as in (3) and (4).

(3) $run(john)$

(4) $\forall x[man(x) \rightarrow run(x)]$.

This gives rise to the question how to obtain rather divergent formulas from closely related sentences. A corresponding question arises for the ambiguity of (5).

(5) *Every man loves a woman.*

This sentence may be used when one specific woman is loved by every man, (say Brigitte Bardot), or when for each man there may be another woman (say his own mother). Sentence (5) is considered as being ambiguous between these two possibilities (for arguments, see section 6). This kind of ambiguity is called 'scope ambiguity' (of quantifiers). The two readings that will be obtained for (5) are (simplified) represented in (6) and (7).

(6) $\forall x[man(x) \rightarrow \exists y[woman(y) \wedge love(x,y)]]$

(7) $\exists y[woman(y) \wedge \forall x[man(x) \rightarrow love(x,y)]]$.

A second kind of phenomena dealt with concerns sentences for which it is difficult to say how their meanings should be represented. Consider (8) and (9)

(8) *John seeks a unicorn.*

(9) *John finds a unicorn.*

These two sentences have about the same form, only the verbs they contain are different. One is tempted to expect that they have about the same meanings as well; the only difference being that they express another relation between John and a unicorn. This is not the case, however. The one sentence gives information about the existence of unicorns, which the other sentence does not. So an approach which says that the *seek*-relation is always a relation between two individuals would not be acceptable. We have to provide a meaning for (8) from which it does not follow that unicorns exist. However, sentence (8) can also be used in a situation that unicorns exist, and it is ambiguous between these two possibilities. It has a reading from which it follows that at least one unicorn exists (the referential reading), and a reading from which this does not follow (the non-referential reading).

Some examples of the referential/non-referential ambiguity are (10), (11), and (12).

(10) *John talks about a unicorn.*

(11) *John wishes to find a unicorn and eat it.*

(12) *Mary believes that John finds a unicorn and that he eats it.*

Sentence (9) allows only for a referential reading. The same holds for sentence (13), see MONTAGUE 1973, p.269.

(13) *John tries to find a unicorn and wishes to eat it.*

The ambiguity we distinguish in sentences (8), (10), (11) and (12) is in the literature also called the '*de-dicto/de-re*' ambiguity, or the '*specific/non-specific*' ambiguity. This terminology is not felicitous, because one might associate with it a distinction that is not covered by the formal analysis that will be provided. Nevertheless, this terminology will sometimes be used in the sequel, since it is standard for some of the examples.

2. JOHN RUNS

The fragment in this section consists of very simple sentences like *John runs*. It has three categories (=sorts): the category T of terms, the category IV of intransitive verb phrases, and the category S of sentences (in PTQ a *t* is used instead of S). There are basic expressions (=generators) of the categories T and IV. The set B_T of generators of the category T contains the proper names of the PTQ-fragment, (B_T ~ 'Basic expressions of category T'). Furthermore a special name is added for illustrative purposes: *Bigboss*. The sets B_T and B_{IV} are defined as follows (B_T will be extended in section 4).

2.1. $B_T = \{John, Bill, Mary, Bigboss\}$

2.2. $B_{IV} = \{run, walk, talk, rise, change\}$.

2.2. END

In the logic there is for each element of B_T a corresponding constant of type e, except for *Bigboss*. In PTQ these constants are called *j,m,b* respectively, but I will use full names: *john* etc.. Notice the difference in the letter type used for English (*Mary*), and the one used for logic (*mary*). One might expect that a proper name translates into the corresponding constant, but for reasons to be explained later, the translation is a complex expression containing this constant. So among the constants in IL of type e, we distinguish three special ones.

2.3 $\{john, bill, mary\} \subset CON_e$.

2.3. END

Constants of type e get as interpretation (with respect to a point of reference) some element in the domain of individuals. This interpretation has to be restricted, for the following reason. If we will speak tomorrow abount John, then we will mean the same individual as today (although he may have some other properties). For instance, if the world would have been different, say, if the Mount Everest would not be the highest mountain, then John would still be the same individual (although his opinion about the Mount Everest might be different). This conception about the individual corresponding with a proper name is expressed by the phrase 'proper names are *rigid designators*'. For an extensive discussion of this conception, see KRIPKE 1972. This idea will be incorporated in our semantics by interpreting constants like *john* 'rigidly', i.e. for each index it will denote the same individual. The name *Bigboss* is to be understood as a surname of the most powerful individual on earth. Since this will not always be the same individual, *Bigboss* is not treated as a rigid designator of type e.

The constants of intensional logic are not interpreted rigidly, on the contrary, they are interpreted index-dependent. I recall the clause for the interpretation of constants:

$$c^{A,i,g} = F(c)(i) \qquad (c \in CON).$$

This means there is no guarantee that the constants corresponding with the proper names of PTQ are interpreted rigidly. Therefore not all possible models for the interpretation of IL are reasonable candidates for an interpretation of English. We will consider only those models in which the constants *john*, *bill*, and *mary* are interpreted rigidly. This is formalized as follows. The requirement of 'rigidity' is expressed by means of an IL-formula, and we will consider only those models in which this formula holds. The formula is called a Meaning Postulate (an MP). It bears index 1 because it is the first meaning postulate in PTQ. Notice that this postulate describes in fact a collection of three formulas.

2.4. Meaning postulate 1:

$$\exists u \Box [u=\alpha] \quad \text{where } \alpha \in \{john, bill, mary\}.$$

2.4. END

This meaning postulate requires that there is one individual in the semantic domain such that the interpretation of *john* equals that individual for all indices. For the PTQ fragment this postulate may be considered sufficient. In more subtle situations this formalization of rigidity is probably too absolute. If epistemological verbs like *know* or *believe* are analysed in detail, then the notion of rigidity may have to be weakened to something like 'in all worlds compatible with the beliefs of some individual such a constant is rigid'. I will, however, follow the PTQ formalization.

An important technical consequence of MP1 is that lambda-conversion is allowed when one of the constants *john*, *bill* or *mary* occurs as argument. First I recall the notation for substitution, for a formal definition see chapter 3, definition 4.3.

2.5. DEFINITION. $[\alpha/z]\phi$ denotes the result of *substitution* of α for all free occurrences of z in ϕ.

2.6. THEOREM.

$$\models \lambda u[\phi](\alpha) = [\alpha/u]\phi$$

where

$$\alpha \in \{john, bill, mary\}.$$

PROOF. MP1 says that for all i,g: $\quad i,g \models \exists u \Box [u=\alpha]$

so there is a $g' \underset{u}{\sim} g$ such that : $\quad i,g' \models \Box [u=\alpha]$

hence for all j : $\quad j,g' \models u = \alpha$

Let i_1 and i_2 be arbitrary. Then:

$$V_{i_1 g}(\alpha) = V_{i_1,g'}(\alpha) = V_{i_1,g'}(u) = g'(u) = V_{i_2,g'}(u) = V_{i_2,g'}(\alpha) = \\ = V_{i_2,g}(\alpha).$$

This means that the condition of theorem 6.4 from chapter 3 is satisfied, hence the theorem allows us to apply λ-conversion.
2.6. END

In the sequel λ-conversion will be used frequently for reducing a formula to a simpler form. Besides λ-conversion several other rules will be introduced for this purpose; they are called reduction rules (RR's). Together they will constitute a procedure which simplifies the formulas obtained by translating the expressions of the fragment. For each reduction rule a correctness proof has to be given, i.e. a proof that the rule transforms a formula into a logically equivalent one. Theorem 6.1 from chapter 3 then allows us to reduce a formula as soon as it is obtained. The purpose of the reduction rules is to obtain formulas which express the intended meaning as clearly and simply as possible. The rules presented in this chapter are almost identical with the rules presented in JANSSEN 1980a. Related reduction rules are discussed in FRIEDMAN & WARREN 1980a,b and INDURKHYA 1981; these authors use the reduction rules for a somewhat different purpose (e.g. to obtain the most extensionalized form), and therefore there are some differences.

The first reduction rule concerns λ-conversion. With respect to this rule the following class of formulas is important: the formulas which contain no operators $^\vee$, H, or W, and which contain as constants only *john, mary* or *bill*. Extending definition 6.2 from chapter 3, I will call these expressions modally closed, since they have the same properties with respect to λ-conversion.

2.7. DEFINITION. An IL formula is called *modally closed* if it is an element of the IL-subalgebra:

$$<[\{john,mary,bill\}],\ (VAR_\tau)_{\tau\in Ty},\ R \cup \{R_\wedge,R_\square\}>$$

where R consists of the operators of Ty2 (recall that $R_\wedge$ and $R_\square$ indicate prefixing with $^\wedge$ and $\square$ respectively).

2.8. Reduction rule 1

Let $z \in \mathrm{VAR}_{\tau_1}$, $\alpha \in \mathrm{ME}_{\tau_2}$, and $\beta \in \mathrm{ME}_{\tau_2}$.
Then replace $\lambda z[\beta](\alpha)$ by $[\alpha/z]\beta$ if
1) no variable in α becomes bound by substitution of α for z in β
and either
2) no occurrence of z in β lies within the scope of $^\wedge$,H,W or $\Box$
or
3) α is modally closed.

CORRECTNESS PROOF

The difference between this rule and theorem 6.3 from chapter 3 is that condition 3 allows for the occurrence of the rigid designators *john, mary* and *bill*. Hence if conditions 1) and 2) are satisfied, the correctness of the λ-conversion follows from that theorem. Suppose now that conditions 1) and 3) are satisfied, and consider the case that α contains of the constants *john*, *bill* and *mary* only occurrences of *john*.

Let w be a variable which does not occur in α or β, and let α' and β' be obtained from α and β by substitution of w for *john*. Consider now

(A) $\lambda w[\lambda z[\beta'](\alpha')](john)$.

Since α' and β' do not contain occurrences of *john* the old conditions for λ-conversion on z are satisfied (chapter 3, theorem 6.3). So (A) is equivalent with:

$$\lambda w[[\alpha'/z]\beta'](john).$$

From theorem 2.6 above, it follows that λ-conversion on w is allowed, so this formula is equivalent with

$$[john/w][[\alpha'/z]\beta'].$$

By the definition of substitution, this is equivalent with

$$[\alpha/z]\beta.$$

So (A) is equivalent with this last formula. On the other hand, we may perform in (A) λ-conversion on *w* because the condition of theorem 2.6 is satisfied. So (A) is also equivalent with

$$\lambda z[\beta](\alpha).$$

The combination of these last two, with (A) equivalent, formulas proves the correctness of λ-conversion for the case that conditions 1) and 3) are satisfied, and that α contains only occurrences of *john*. For other constants and for occurrences of more than one constant, the proof proceeds analogously.

2.8. END

As said before, at different indices different persons can be Bigboss. Therefore we cannot translate *Bigboss* into a rigid constant of type e. We might translate it into a constant of type <s,e>, or into a constant of type e and interpret it non-rigidly. I choose the former approach (thus being consistent with the examples involving *bigboss* given in section 7 of chapter 3). This explains the following definition

2.9 $\quad$ *bigboss* $\in CON_{<s,e>}$.

2.9. END

The interpretation of the constant *bigboss* is a function from indices to individuals. Such a function is called an *individual concept*. Also $^\wedge$*john* denotes an individual concept. The individual concept denoted by $^\wedge$*john* is a constant function, whereas the one denoted by *bigboss* is not. One might expect that *Bigboss* translates into the corresponding constant. But, as for the other proper names, it will be explained later why this is not the case.

Suppose that the balance of power changes and Bresjnev becomes Bigboss instead of Reagan. Then this might be expressed by sentence (14).

(14) *Bigboss changes.*

The meaning of (14) is not correctly represented by a formula which says that the predicate *change* applies to a certain individual. Who would that be? Maybe there was a change in the absolute power of Reagan (it decreased), or in the absolute power of Bresjnev (it increased). Probably both persons changed with respect to power. Sentence (14) rather says that the concept

'Bigboss' has changed in the sense that it concerns another person. So the meaning of (14) can be represented by a formula which says that the predicate *change* holds for the individual concept related with *Bigboss*. In such an analysis *change* has to be of type <<s,e>,t>. Due to the homomorphic relation between syntax and semantics, this means that all intransitive verbs have to be of type <<s,e>,t>.

At this stage of the description of the fragment the only example of an argument of type <s,e> is the artificial example *bigboss*. In appendix 2 of this book, other examples will be given where the translation of the argument of a property has to be of this type. This discussion explains the introduction of the following constants and translations. The translation function is indicated by means of a '(prime). Note that this is a different use of ' than in PTQ (there it distinguishes English words from logical constants).

2.10 $\{run, walk, talk, rise, change\} \subset \mathrm{CON}_{<<s,e>,t>}$

2.11 $run' = run$, $walk' = walk$, $talk' = talk$
$rise' = rise$, $change' = change$.

2.11. END

One might be tempted to take the constant *john* as translation of the proper name *John*. In the fragment consisting only of sentences like *John runs* there would be no problem in doing so. But there are more terms, and the similarity of syntax and semantics requires that all terms are translated into expressions of the same type. We already met the proper name *Bigboss*, translating into an expression of type <s,e>. One might expect $^\wedge$*john* as translation for *John*. But in the sequel we will meet more terms: e.g. *every man*. If we would translate *John* into an expression denoting an individual concept (or alternatively an individual), then *every man* has to be translated into such an expression as well. Would that be possible?

The idea is discussed by LEWIS (1970). He tells us that in the dark ages of logic a story like the following was told. 'The phrase every pig names a [..] strange thing, called the universally generic pig, which has just those properties that every pig has. Since not every pig is dark, pink, grey or of another color, the universally generic pig is not of an any color (Yet neither he is colorless, since not every - indeed not any - pig is colorless)'. (LEWIS 1970, p.35). This illustrates that this approach is not sound. Therefore, we will forget the idea of universal generic objects (for a proposal for a reconstruction, see Van BENTHEM 1981a), and we will interpret the term *every man* as the set of properties every man has. As a consequence of the similarity of syntax and semantics, all other terms will denote sets of properties as well.

On the basis of this argumentation one might expect for *John* the translation $\lambda z[z(^{\wedge}john)]$, where z is a variable of type <<s,e>,t>. But this is not adequate for the following reason. A variable of type <<s,e>,t> denotes (the characteristic function of) a set of individual concepts. What we usually take to be a property cannot be adequately formalized in this way. Consider the property 'being a football player'. This would be formalized as a set of individual concepts. The same holds for the property of 'being a member of the football union': this is formalized as a set of individual concepts as well. Suppose now that (for a certain index) all football players are members of the football union. Then these two sets would be the same, so the two properties would be formalized in the same way. But we do not consider these two properties as being the same. In other circumstances (for other indices) there might be players who are not a member of the union. In order to formalize these differences, properties are taken to be of one intensional level higher hence a variable which ranges over properties has to be of type <s,<<s,e>,t>>. This explains the following translations of proper names.

2.12. <u>Translations</u>

$John' = \lambda P[[^{\vee}P](^{\wedge}john)]$, $Bill' = \lambda P[[^{\vee}P](^{\wedge}bill)]$

$Mary' = \lambda P[[^{\vee}P](^{\wedge}mary)]$, $Bigboss' = \lambda P[[^{\vee}P](bigboss)]$

here $P \in \text{VAR}_{<s,<<s,e>,t>>}$.

2.12. END

After this discussion concerning the proper names and intransitive verbs, the rule for their combination can be given. I first quote the PTQ formulation, since this way of presentation is in the literature the standard one. The formulation of the rule contains expressions like '$\alpha \in P_T$', this should be read as 'α is a phrase of the category T'. The rule is called S_4, because it is the fourth syntactic rule of PTQ, and I wish to follow that numbering when possible.

2.13. <u>Rule S_4</u>

If $\alpha \in P_T$ and $\beta \in P_{IV}$ then $F_4(\alpha,\beta) \in P_S$, where $F_4(\alpha,\beta) = \alpha\tilde{\beta}$ and $\tilde{\beta}$ is the result of replacing the first verb in β by its third person singular present.

2.13. END

This formulation of the rule contains a lot of redundancy, and therefore I will use a more concise presentation. As one remembers from the previous chapters, the syntactic rules are operators in an algebraic grammar. The form of representation I will use, resembles closely the representations used in the previous chapters for algebraic operators. First it will be said what kind of function the rule is; as for S_4 it is a function from $T \times IV$ to S (written as $T \times IV \to S$). Next it will be described how the effect of the operator is obtained. I will use a notation that suggests that some basic operations on strings are available, in particular a concatenation operator which yields the concatenation of two strings as result. The semi-colon (;) is used to separate the consecutive stages of the description of the syntactic operator; it could be read as 'and next'. Furthermore the convention is used that α always denotes the expression which was the first argument of the syntactic rule. If this expression is changed in some step of the syntactic operation, it will then denote the thus changed expression. For the second argument β is used in the same way. Rule S_4 presented in this format reads as follows.

2.14. <u>Rule S_4</u>

$T \times IV \to S$

F_4: replace the first verb in β by its third person singular present; concatenate (α,β).

2.14. END

The occurrence of the name F_4 is a relict of the PTQ formulation, and might be omitted here. But in a context of a long list of rules it is sometimes useful to have a name for an operation on strings, because it can then be used in the description of other rules.

The translation rule corresponding with S_4 reads in PTQ as follows.

2.15. T_4:

If $\alpha \in P_T$, $\beta \in P_{IV}$, and α,β translate into α',β' respectively, then $F_4(\alpha,\beta)$ translates into $\alpha'(^\wedge\beta')$.

2.15. END

Also the translation rule contains a lot of redundant information. Let us denote by α' the translation of the first, and by β' the translation of the second argument of the preceding syntactic rule. Then a translation rule can fully be described by giving just the relevant logical expression (polynomial over IL with α' and β' as parameters). What the types of α' and β' are, follows immediately from the sorts mentioned in the syntactic rule T_4 presented in this format reads:

2.16. T_4:

$\alpha'(^\wedge\beta')$

2.16. END

Now we come to the production of sentence (15), viz. *Bigboss changes*. This sentence, containing the artificial term *Bigboss*, is given as the first example because all information needed for a full treatment of this sentence is given now; sentences like *John changes* have to wait for a moment. Sentence (15) is obtained by application of S_4 to the basic term *Bigboss* and the basic verb *change*. This information is presented in the tree in figure 1. The S in {S,4} stands for the category of the obtained expression, the 4 for the number of the rule used to produce the expression

(15) *Bigboss changes*.

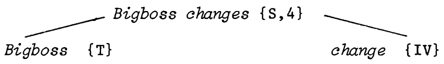


Figure 1

The translation of *Bigboss* is $\lambda P[^\vee P(bigboss)]$, and the translation of *change* is *change*. If we combine *Bigboss* and *change* according to rule S_4, thus producing (15), then the translation of the result is obtained by application of T_4 to their respective translations. Since

$T_4(\alpha',\beta') = \alpha'({}^\wedge\beta')$, sentence (15) translates into (16).

(16) $\lambda P[{}^\vee P(\mathit{bigboss})]({}^\wedge \mathit{change})$.

Now conditions 1 and 2 of reduction rule RR_1 are satisfied. So this formula can be reduced to (17).

(17) $[{}^{\vee\wedge}\mathit{change}](\mathit{bigboss})$.

This formula can be simplified further using the following reduction rule.

2.17. Reduction Rule 2

Let be given a formula of the form ${}^{\vee\wedge}\alpha$. Then replace this formula by α.

CORRECTNESS PROOF. $\models {}^{\vee\wedge}\alpha = \alpha$ see chapter 3, theorem 7.1.
2.17. END

Using this reduction rule formula (17) reduces to (18).

(18) *change*(*bigboss*).

This formula expresses that the predicate *change* holds for the individual concept *bigboss*.

Instead of all this verbosity, we might present the translations immediately in the tree. Depending on the complexity of the formulas involved, these may be unreduced, partially reduced or completely reduced formulas. An example is given in figure 2.

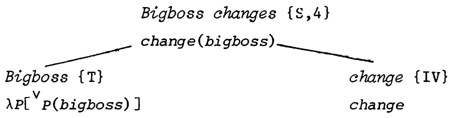


Figure 2

Another method to present the production and translation process is to write this in an algebraic way, of which the following is an example.

$[\mathit{Bigboss\ changes}]' = [S_4(\mathit{Bigboss},\mathit{change})]' =$
$= T_4(\mathit{Bigboss}',\mathit{change}') = \mathit{Bigboss}'({}^\wedge\mathit{change}') =$
$= [\lambda P[{}^\vee P(\mathit{bigboss})]({}^\wedge\mathit{change})] = \{RR_1\} = [{}^{\vee\wedge}\mathit{change}](\mathit{bigboss}) = \{RR_2\} =$
$= \mathit{change}(\mathit{bigboss})$

The treatment of *Mary walks* proceeds, in its first stage, analogously to the treatment of *Bigboss changes*, see figure 3.

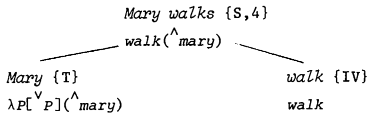


Figure 3

The formula obtained as translation for *Mary walks*, is not completely satisfactory. Intuitively one interprets this sentence as stating that a certain predicate (denoting the property of walking) holds for a certain individual (Mary). This is not reflected in the obtained translation; in $walk(^\wedge mary)$ a predicate is applied to an individual concept. Since *mary* is a rigid constant, $^\wedge mary$ denotes a function which yields for all indices the same individual. Saying that this constant function has a certain property is tantamount to saying that the corresponding individual has a certain property (there is a 1-1 correspondence between individuals and functions yielding always the same individual). However, one would like to have reflected in the translation of *Mary walks* that a predicate holds for an individual. Therefore the following notation is introduced (see PTQ, p.265).

2.18. DEFINITION. Let $\delta \in CON_{<<s,e>,t>}$. Then δ_* is an abbreviation for $\lambda u\delta(^\wedge u)$ (so $\delta_* \in ME_{<e,t>}$).
2.18. END

Consequently we have the following rule for simplifying formulas.

2.19. Reduction rule 3

Let be given a formula of the form $\delta(^\wedge\alpha)$, where $\delta \in CON_{<<s,e>,t>}$ and $\alpha \in VAR_e$ or $\alpha \in \{john, bill, mary\}$. Then replace $\delta(^\wedge\alpha)$ by $\delta_*(\alpha)$.

CORRECTNESS PROOF. $\delta_*(\alpha) = \lambda u[\delta(^\wedge u)](\alpha) =\{RR_1\}= \delta(^\wedge\alpha)$. Note that λ-conversion is allowed because the mentioned constants of type e are rigid designators.
2.19. END

Using RR3 the translation of *Mary walks* in figure 3, reduces to (19).

(19) $walk_*(mary)$.

As last example I present the treatment of the sentence mentioned in the title of this section. For variation I use not the tree representation, but the algebraic one.

$[John\ runs]' = [S_4(John,run)]' = John'(^\wedge run') = \lambda P[[^\vee P](^\wedge john)](^\wedge run) = \{RR_1\} = [^{\vee\wedge} run](^\wedge john) = \{RR_2\} = run(^\wedge john) = \{RR_3\} = run_*(john)$.

In PTQ more is said about the fragment presented so far. A meaning postulate (MP_3) is introduced which says that the truth of e.g. $walk(x)$ only depends on the extension of x, i.e. the subject position of walk is extensional. In appendix 2 of this book the problems of extension and intension will be discussed, and this postulate will be considered. For verbs of other categories the extensionality of the subject position is guaranteed by meaning-postulates as well, (see appendix 1).

3. THE WOMAN WALKS

In this section the fragment is extended with the categories of Common Nouns (CN) and of determiners (Det). The treatment of determiners given here differs from their PTQ treatment. In PTQ determiners are introduced syncategorematically, introducing each determiner by a distinct rule. Maybe the motivation for Montague to do so, was that in logic quantifiers are usually introduced syncategorematically. From a linguistic point of view it is more attractive to have determiners in a separate category (they form a group of expressions which behave syntactically in a regular way). Since I do not know any argument against treating them categorially, the PTQ approach is not followed here. The generators of the two new categories are as follows

3.1. $B_{CN} = \{man, woman, park, fish, pen, unicorn, price, temperature\}$

3.2. $B_{Det} = \{every, a, the\}$

3.2. END

For each element in B_{CN} there is a corresponding constant, and the common nouns translate into these constants. The nouns are treated semantically in the same way as the intransitive verbs we have met in section 2. Hence the nouns translate into constants of type <<s,e>,t>. This explains the following definitions.

3.3. $\{man, woman, park, fish, pen, unicorn, price, temperature\} \subset \mathrm{CON}_{<<s,e>,t>}$

3.4. *man'* = *man*, *woman'* = *woman*, *park'* = *park*, *pen'* = *pen*,
unicorn' = *unicorn*, *price'* = *price*, *temperature'* = *temperature*.

3.4. END

An example of a formula containing the constant *bill* is (20), in which is expressed that Bill is a man.

(20) $man(^{\wedge}bill)$.

The δ_{*}-notation (definition 2.18) is applicable to all constants of type <<s,e>,t>, so it can be applied to constants translating common nouns as well. So (20) may be replaced by (21).

(21) $man_{*}(bill)$.

Out of a CN and a determiner a term can be formed, using the following rule.

3.5. Rule S_2

Det × CN → T

F_2: concatenate(α,β)
T_2: $\alpha'(^{\wedge}\beta')$.

Example

F_2(*a*,*woman*) = *a woman*.

3.5. END

We wish to use the terms produced with this rule in the same way as we used the term *John*: rule S_4 should be applicable to the result of S_2, yielding sentences like (22), (23) and (24).

(22) *A woman runs*
(23) *Every woman runs*
(24) *The woman runs*.

The meanings associated with determiners are best understood by considering the meanings that we wish to assign to the above sentences (cf. the discussion concerning contextuality and compositionality in section 2 of chapter 1). Let us accept (for the moment without explanation) the quantification over individual concepts; then the translations of (22),

(23) and (24) are (25), (26) and (27) respectively.

(25) $\exists x[woman(x) \wedge run(x)]$

(26) $\forall x[woman(x) \rightarrow run(x)]$

(27) $\exists x \forall y[[woman(y) \leftrightarrow x=y] \wedge run(x)]$.

The last formula is somewhat complex. It says that there is an entity x which is a woman, and that for any entity y which is a woman holds that it is identical to the entity x. In other words, (27) is false when there is no woman at all, and it is false when there is more than one woman. This kind of analysis for *the* is called the Russellian analysis, because it was proposed by Russell to deal with the famous example (28).

(28) *The present King of France is bald.*

The meanings of the terms have to be such that if they take an IV-translation as argument, the resulting translations are the ones we desired for the obtained sentences. Hence their translations have to be of the same kind as the translation of the term *John*: a (characteristic function of a) set of properties of an individual concept. So we wish to translate (29) by (30).

(29) *a woman*

(30) $\lambda P \exists x[woman(x) \wedge {}^{\vee}P(x)]$.

Formula (30) is interpreted as the characteristic function of those properties P such that there is at least one woman which has this property P. Other determiners are treated analogously. As translation for the determiner *a* we take formula (30), but with *woman* replaced by a variable. This variable is of type <s,<<s,e>,t>> (the reason for this is the same as the reason given for the type of the variable P, see the translation of *John*). This explains the following translations of determiners.

3.6. Translations of determiners

$$\begin{aligned} every' &= \lambda Q \lambda P \forall x[{}^{\vee}Q(x) \rightarrow {}^{\vee}P(x)] \\ a' &= \lambda Q \lambda P \exists x[{}^{\vee}Q(x) \wedge {}^{\vee}P(x)] \\ the' &= \lambda Q \lambda P \exists x[\forall y[{}^{\vee}Q(y) \leftrightarrow x=y] \wedge {}^{\vee}P(x)]. \end{aligned}$$

3.6. END

Formulas (25), (26) and (27) are not in all respects a satisfactory representation of the meanings of sentences (22), (23) and (24) respectively. The formulas contain quantifications over individual concepts, whereas one would prefer a quantification over individuals. The conditions for application of RR_3 are not satisfied, so we have no ground for the elimination of the individual concepts by means of an application of this rule. On the contrary: as I will explain, the replacement of (31) by (32) would replace a formula by a non-equivalent one.

(31) $\exists x[woman(x) \wedge run(x)]$

(32) $\exists u[woman_*(u) \wedge run_*(u)]$.

A possible choice for the value of *x* in (31) would be to assign to *x* the same interpretation as to *bigboss*, but in (32) there is not a corresponding choice. One would prefer to have (32) as the meaning representation of the meaning of (25) because intuitively (25) gives information about individuals, and not about individual concepts. Following Montague, we obtain this effect by means of the introduction of a meaning postulate. Only those models for intensional logic are possible models for the interpretation of English in which the following meaning postulate holds.

3.7. <u>Meaning postulate 2</u>

$$\Box\,[\delta(x) \rightarrow \exists u[x={}^{\wedge}u]]$$

where $\delta \in \{man, woman, park, fish, pen, unicorn\}$.

3.7. END

This meaning postulate says that constants such as *man* can yield true only for constant individual concepts, i.e. for individual concepts which yield for every index the same individual. Note that the constants *price* and *temperature* are not mentioned in MP_2. Arguments for this, and examples involving *price* and *temperature* will be given in appendix 1 of this volume. As a consequence of MP_2, it can be shown that (31) and (32) are equivalent. I will not present a proof for this, because it is only one of the situations in which MP_2 will be used. In appendix 1, it will be investigated in general in which circumstances MP_2 allows us to replace a quantification over individual concepts by a quantification over individuals. For the moment it suffices to know that in all examples we will meet, such a replacement is allowed. This is expressed in the following reduction rule.

3.8. <u>Reduction Rule 4</u>

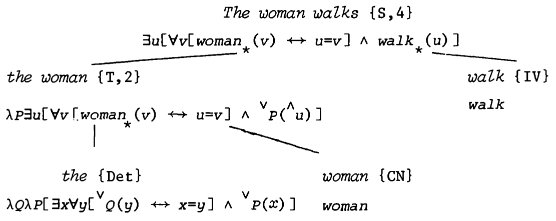
Let be given a formula of one of the following forms: $\exists x[\delta(x) \wedge \phi(x)]$, $\forall x[\delta(x) \rightarrow \phi(x)]$ or $\exists x[\forall y[\delta(y) \leftrightarrow x=y] \wedge \phi(x)]$.

If MP2 holds for δ, then replace these formulas by respectively $\exists u[\delta(^\wedge u) \wedge \phi(^\wedge u)]$, $\forall u[\delta(^\wedge u) \rightarrow \phi(^\wedge u)]$ or $\exists u[\forall v[\delta(^\wedge v) \leftrightarrow u=v] \wedge \phi(^\wedge u)]$.

<u>CORRECTNESS PROOF</u>. See appendix 2.

3.8. END

The production of the sentence mentioned in the title of this section is given in figure 4.



The woman walks {S,4}
$\exists u[\forall v[woman_*(v) \leftrightarrow u=v] \wedge walk_*(u)]$
the woman {T,2}
$\lambda P\exists u[\forall v[woman_*(v) \leftrightarrow u=v] \wedge {}^\vee P(^\wedge u)]$
walk {IV}
walk
the {Det}
$\lambda Q\lambda P[\exists x\forall y[{}^\vee Q(y) \leftrightarrow x=y] \wedge {}^\vee P(x)]$
woman {CN}
woman

<u>Figure 4</u>

Note how in this simple example RR_4 and RR_3 are used in order to simplify the translation of *the woman*, and RR_1 and RR_3 to simplify the translation of *the woman walks*. In the sequel such reductions will often be performed without any further comment.

4. MARY WALKS AND SHE TALKS

In this section the fragment is extended with rules for disjunction and conjunction, and with a rule for co-referentiality. The rules for producing conjoined sentences are as follows.

4.1. <u>Rule S_{11a}:</u>

$S \times S \rightarrow S$

F_{11a}: concatenate (α, *and*, β)

T_{11a}: $\alpha' \wedge \beta'$

4.2. Rule S_{11b}:

$S \times S \to S$

F_{11b}: concatenate (α, *or*, β)
T_{11b}: $\alpha' \vee \beta'$.

4.2. END

Notice that the words *and* and *or* are not members of a category of connectives: they are introduced syncategorematically. It would be possible to have a three-place rule for sentence conjunction, with for the connective *and* as translation $\lambda\phi\lambda\psi[\phi \wedge \psi]$. This categorical approach is not followed here because there are rules for disjunction and conjunction for other categories as well. Furthermore, the situation is complicated by the fact that there is term disjunction in the fragment, but no term conjunction (in order to avoid plurals). In this situation it would not be a simplification to use a categorical treatment of connectives. For a categorical treatment in a somewhat different framework, see GAZDAR 1980.

The rules for forming conjoined phrases of other categories than sentences are as follows.

4.3. Rule S_{12a}:

$IV \times IV \to IV$

F_{12a}: concatenate (α, *and* β)
T_{12a}: $\lambda x[\alpha'(x) \wedge \beta'(x)]$.

4.4. Rule S_{12b}:

$IV \times IV \to IV$

F_{12b}: concatenate (α, *or* β)
T_{12b}: $\lambda x[\alpha'(x) \vee \beta'(x)]$.

4.5. Rule S_{13}:

$T \times T \to T$

F_{13}: concatenate (α, *or* β)
T_{13}: $\lambda P[\alpha'(P) \vee \beta'(P)]$.

4.5. END

The production of (33) is given in figure 5.

(33) *John walks and talks.*

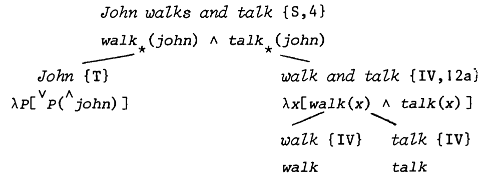

John walks and talk {S,4}
$walk_*(john) \wedge talk_*(john)$
John {T}
$\lambda P[{}^\vee P({}^\wedge john)]$
walk and talk {IV,12a}
$\lambda x[walk(x) \wedge talk(x)]$
walk {IV}
walk
talk {IV}
talk

Figure 5

Note that the produced sentence is not identical with (33). The treatment presented in figure 5 obeys the formulation of S_4, and, therefore, only the first verb is conjugated. For an improved treatment see chapter 8, or FRIEDMAN 1979b.

An example of term disjunction is given in (34).

(34) *John or Mary talks.*

First (35) is formed according to S_{13}. Its unreduced translation is (36).

(35) *John or Mary*

(36) $\lambda P[\lambda P[{}^\vee P({}^\wedge john)](P) \vee \lambda P[{}^\vee P({}^\wedge mary)](P)]$.

Formula (36) contains several occurrences of the variable P, and three binders for P (viz. three occurrences of λP). However, due to the different scopes of the lambda operators, it is uniquely determined which variables occur in the scope of each of the lambda operators. The conditions for λ-conversion are satisfied, and after two applications of RR_1 formula (36) reduces to (37).

(37) $\lambda P[{}^\vee P({}^\wedge john) \vee {}^\vee P({}^\wedge mary)]$.

Application of S_4 to term (35) and the verb *talk*, yields (34), which has as unreduced translation (38). This formula reduces by application of RR_1 and RR_2 to (39), and using RR_3 to (40).

(38) $\lambda P[{}^\vee P({}^\wedge john) \vee {}^\vee P({}^\wedge mary)]({}^\wedge talk)$

(39) $talk({}^\wedge john) \vee talk({}^\wedge mary)$

(40) $talk_*(john) \vee talk_*(mary)$.

In sentences containing conjunctions or disjunctions pronouns occur often which are coreferential with some other term in that sentence. An example is the coreferentiality of *she* and *Mary* in (41).

(41) *Mary walks and she talks.*

In order to account for coreferentiality, a collection of new -artificial- terms is introduced. Since they have a relationship with logical variables, they are called syntactic variables. These variables are not words of English, and might be represented by means of some artificial symbol. Since the variables are related to pronouns, it has some advantages, to give them a representation exhibiting this relationship. The variables are written as male pronouns provided with an index (e.g. he_n). Their translations contain logical variables x_n of type <s,e>. The syntactic variables he_n are generators of sort T.

4.6. $\{he_1, he_2, \ldots\} \subset B_T$.

4.7. $he_1' = \lambda P[{}^{\vee}P(x_1)]$, $he_2' = \lambda P[{}^{\vee}P(x_2)], \ldots$.

4.7. END

One of the most important rules of PTQ is S_{14}. As for the syntax it removes the syntactic variables. As for the translation, it binds the corresponding logical variables. This rule enables us to deal with most of the ambiguities mentioned in the introduction, but in this section we will only deal with its use for coreferentiality. In fact S_{14} is not a rule, but rather a rule-scheme which for each choice of the index n constitutes a rule. This aspect will be indicated by using the parameter n in the description of the rule scheme.

4.8. <u>Rule $S_{14,n}$:</u>

$T \times S \rightarrow S$

$F_{14,n}$: If $\alpha = he_k$ then replace all occurrences of he_n/him_n in β by he_k/him_k respectively.
Otherwise replace the first occurrence of he_n in β by α, and replace all other occurrences of he_n in β by *he/she/it* and of him_n by *him/her/it* according to the gender of the first CN or T in α.

$T_{14,n}$: $\alpha'(^\wedge\lambda x_n[\beta'])$.

4.8. END

An example of the use of (an instance of) $S_{14,n}$ arises in the production of (41), as presented in figure 6.

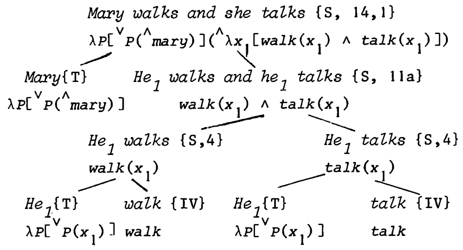

Mary walks and she talks {S, 14,1}
$\lambda P[^\vee P(^\wedge mary)](^\wedge\lambda x_1[walk(x_1) \wedge talk(x_1)])$
Mary{T}
$\lambda P[^\vee P(^\wedge mary)]$
He₁ walks and he₁ talks {S, 11a}
$walk(x_1) \wedge talk(x_1)$
He₁ walks {S,4}
$walk(x_1)$
He₁ talks {S,4}
$talk(x_1)$
He₁{T}
$\lambda P[^\vee P(x_1)]$
walk {IV}
walk
He₁{T}
$\lambda P[^\vee P(x_1)]$
talk {IV}
talk

Figure 6

The translation for (41) given in figure 6 can be reduced, using RR3, to (42).

(42) $[^{\wedge\vee}\lambda x_1[walk(x_1) \wedge talk(x_1)]](^\wedge mary)$.

By application of RR2 and RR1 this reduces to (43), and by RR3, further to (44).

(43) $walk(^\wedge mary) \wedge talk(^\wedge mary)$

(44) $walk_*(mary) \wedge talk_*(mary)$.

Some syntactic details of $S_{14,n}$ give rise to problems. The rule for term disjunction allows us to produce term phrases like *he$_1$ and Mary*, and *he$_1$ or he$_2$*. In both cases it is not clear what is to be understood by the gender of the first T or CN in such a term. And if the term *John or Mary* is formed, it is not correct to use the pronoun *he*, but one should use *he or she*, witness the following example (FRIEDMAN, 1979).

(45) *John or Mary walks and he or she talks.*

It would require a more sophisiticated syntax than we have available here in order to account correctly for these problems (see FRIEDMAN 1979 for an improved treatment).

The detail of $S_{14,n}$ that the first occurrence of *he$_n$/him$_n$* has to be replaced, is explained as follows. A pronoun may always be coreferential with a common noun or term occurring earlier in the sentence, but it may not always refer forward to terms or nouns occurring later. So it is a safe strategy to put the coreferential noun phrase always in a position which is as leftmost as possible. It is a difficult, and not completely solved task, to characterize the situations in which a pronoun may refer to a term occurring later in the sentence. Therefore S_{14} describes only reference to terms occurring earlier than the pronoun. Even this safe procedure does not avoid all problems. In some cases a personal pronoun is produced, where a reflexive pronoun is required. Sentence (46) has, according to the rules described here, a translation which expresses that John loves himself. This result is, of course, incorrect.

(46) *John loves him.*

Our aim was to deal with certain semantic problems, and therefore I will not consider here proposals for dealing with this syntactic problem (one of the proposals from the literature, viz. PARTEE 1973, will be considered in chapters 5 and 6 although not from the present point of view).

5. JOHN FINDS A UNICORN

In this section the category TV of transitive verb phrases is introduced. The generators of this category are as follows.

5.1. $B_{TV} = \{\textit{find},\textit{loose},\textit{eat},\textit{love},\textit{date},\textit{be},\textit{seek},\textit{conceive}\}$.

5.1. END

Corresponding with these TV's (except for *be*), there are constants in the logic. They denote higher order functions which take as argument the intension of a term translation, and yield an element of the same type as the translations of IV-phrases. The translations of the basic verbs of the category TV are the corresponding constants; the translation of *be* is a compound expression of the same type. Let us indicate by $\tau(C)$ the type of the translation of an expression of category C. Then $\tau(TV) = \langle\langle s,\tau(T)\rangle,\tau(IV)\rangle$. This explains the following definitions

5.2. $\{\textit{find},\textit{loose},\textit{eat},\textit{love},\textit{date},\textit{seek},\textit{conceive}\} \subset CON_{\langle\langle s,\tau(T)\rangle,\tau(IV)\rangle}$

5.3. $\mathit{find}' = \mathit{find}$, $\mathit{loose}' = \mathit{loose}$, $\mathit{eat}' = \mathit{eat}$, $\mathit{love}' = \mathit{love}$,
$\mathit{be}' = \lambda P \lambda x[{}^{\vee}P({}^{\wedge}\lambda y[{}^{\vee}x = {}^{\vee}y])]$ where $P \in \mathrm{VAR}_{<s,\tau(T)>}$
$\mathit{seek}' = \mathit{seek}$, $\mathit{conceive}' = \mathit{conceive}$.

5.3. END

Out of a TV and a Term and IV can be formed according to the following rule.

5.4. Rule S_5:

$TV \times T \rightarrow IV$

F_5: concatenate (α,β)
T_5: $\alpha'({}^{\wedge}\beta')$.

5.4. END

An example of the use of this rule is the production of (47), partially presented in figure 7.

(47) *John seeks a unicorn.*

$$\begin{tikzcd}[column sep=small]
 & \begin{array}{l}\textit{John seeks a unicorn}\ \{S,4\}\\ \mathit{seek}({}^{\wedge}\lambda P\exists u[\mathit{unicorn}_{*}(u) \wedge {}^{\vee}P({}^{\wedge}u)])({}^{\wedge}\mathit{john})\end{array} \arrow[dl, dash] \arrow[dr, dash] & & \\
\begin{array}{l}\textit{John}\ \{T\}\\ \lambda P[{}^{\vee}P({}^{\wedge}\mathit{john})]\end{array} & & \begin{array}{l}\textit{seek a unicorn}\ \{IV,5\}\\ \mathit{seek}({}^{\wedge}\lambda P\exists u[\mathit{unicorn}_{*}(u) \wedge {}^{\vee}P({}^{\wedge}u)])\end{array} \arrow[dl, dash] \arrow[dr, dash] & \\
 & \begin{array}{l}\textit{seek}\ \{TV\}\\ \mathit{seek}\end{array} & & \begin{array}{l}\textit{a unicorn}\ \{T,2\}\\ \lambda P\exists u[\mathit{unicorn}_{*}(u) \wedge {}^{\vee}P({}^{\wedge}u)]\end{array}
\end{tikzcd}$$

Figure 7

The translation obtained in figure 7 is not the traditional one: one would like to consider *seek* as a two-place relation. Therefore the following convention is introduced.

5.5. DEFINITION. $\gamma(\alpha,\beta) = \gamma(\beta)(\alpha)$, where γ is an expression translating a TV.
5.5. END

In PTQ (p.259) this convention is defined for all γ. It is however only useful for TV's (see section 11). The above definition gives rise to the following reduction rule.

5.6. Reduction rule 5

Let be given a formula of the form $\gamma(\beta)(\alpha)$, where γ is the translation of some transitive verb. Then replace this formula by $\gamma(\alpha,\beta)$.

CORRECTNESS PROOF

See definition 5.5.

5.6. END

Using RR5, the formula obtained in figure 7 reduces to (48).

(48) $seek({}^{\wedge}john,\ {}^{\wedge}\lambda P\exists u[unicorn_{*}(u) \wedge {}^{\vee}P({}^{\wedge}u)])$.

This translation describes the de-dicto reading of (47). The de-re reading will be considered in section 6. Below I will discuss whether the formula expresses a relation between the right kinds of semantic objects.

The first argument of *seek* is a constant individual concept. One might wish to have an individual as first argument. In analogy of the δ_{*} notation for intransitive verbs, we might introduce a notation for transitive verbs in which the $^{\wedge}$ in front of *john* disappears. PARTEE (1975, p.290) has proposed such a notation, but it is not employed in the literature, therefore I will not use it here. Notice that the interpretation of (48) is tantamount to a relation of which the first component is an individual (see section 2).

The second argument in (48) is the intension of a collection of properties. So *seek* is not treated as a relation between two individuals, and therefore (48) does not allow for the conclusion that there is a particular unicorn which John seeks. In this way the problem mentioned in section 1 is solved, so in this respect the formula is satisfactory. But one might ask whether this effect could be obtained by means of a simpler formula, viz. one without the intension sign. The need for this intension in the second argument is explained as follows (JANSSEN 1978b, p.134). Suppose that *seek* is considered as a relation between an individual and (the characteristic function of) a set of properties. Consider a world in which there exist no unicorns. Then for no property P it is true that $\exists u[unicorn_{*}(u) \wedge {}^{\vee}P({}^{\wedge}u)]$. Thus in these circumstances $\lambda P\exists u[unicorn_{*}(u) \wedge {}^{\vee}P({}^{\wedge}u)]$ is the characteristic function of the empty set of properties. The semantic interpretation of *John seeks a unicorn* then states that the seek-relation holds between John and this empty set. Suppose moreover that in this world also no centaurs exist. Then the semantic interpretation of

(49) *John seeks a centaur*

also expresses that the seek-relation holds between John and the empty set of properties. But this contradicts our intuition that (47) and (49) have different meanings. When we wish to describe the difference between centaurs and unicorns we cannot restrict our attention to the present state of the present world. We should also consider other worlds (or other states of the present world) for instance, those in which unicorns or centaurs do exist. In other worlds the set $\lambda P\exists u[unicorn_*(u) \wedge {}^{\vee}P({}^{\wedge}u)]$ might be different from $\lambda P\exists u[centaur_*(u) \wedge {}^{\vee}P({}^{\wedge}u)]$. Therefore the *seek*-relation will be considered as a relation between individuals and intensions of sets of properties. Since these intensions are different, *seek a unicorn* will get an interpretation different from the one for *seek a centaur* (even if both are extinct).

In the same way as we produced *John seeks a unicorn*, we may produce (50) with as reduced translation (51).

(50) *John seeks Mary*

(51) $seek({}^{\wedge}john, {}^{\wedge}\lambda P[{}^{\vee}P]({}^{\wedge}mary))$.

This formula expresses that the seek relation holds between an individual concept and the collection of properties of Mary. But sentence (50) expresses that the seek-relation holds between two individuals: between John and Mary. One would like to have this aspect expressed by the obtained formula. Therefore the following definition (PTQ, p.265).

5.7. <u>DEFINITION</u>. $\delta_* = \lambda v\lambda u\delta({}^{\wedge}u, {}^{\wedge}\lambda P[{}^{\vee}P({}^{\wedge}v)])$, where $\delta \in CON_{\tau(TV)}$.
5.7. END

On the basis of this definition we have the following reduction rule.

5.8. <u>Reduction rule 6</u>

Let be given an expression of the form $\delta({}^{\wedge}\alpha, {}^{\wedge}\lambda P[{}^{\vee}P({}^{\wedge}\beta)])$, where $\alpha,\beta \in VAR_e \cup CON_e$, and $\delta \in CON_{\tau(TV)}$. Then replace this expression by $\delta_*(\alpha,\beta)$.

CORRECTNESS PROOF

$\delta_*(\alpha,\beta) = \delta_*(\beta)(\alpha) = \lambda v \lambda u \delta(^\wedge u, ^\wedge\lambda P[^\vee P(^\wedge v)])(\beta)(\alpha) = \{RR1\} =$
$= \delta(^\wedge\alpha, ^\wedge\lambda P[^\vee P(^\wedge\beta)])$.

Note that λ-reduction is allowed because the constants of type e in the fragment are rigid.

5.8. END

Using RR6 we may reduce (51) to (52).

(52) $seek_*(john,mary)$.

In the same way as we produced the sentence *John seeks a unicorn*, we may produce (53), with translation (54).

(53) *John finds a unicorn*

(54) $find(^\wedge john, ^\wedge\lambda P[\exists u\ unicorn_*(u) \wedge {}^\vee P(^\wedge u)])$.

This result is not precisely what we would like to have. Sentence (53) gives the information that there exists at least one unicorn, and (54) does not express this information. In order to deal with this aspect we restrict our attention to those models for IL in which the following meaning postulate is satisfied.

5.8. Meaning Postulate 4

$$\exists S \forall x \forall P \Box\, [\delta(x,P) \leftrightarrow {}^\vee P(^\wedge\lambda y\, {}^\vee S(^\vee x, {}^\vee y))]$$

where $\delta \in \{find, loose, eat, love, date\}$ and $P \in VAR_{<s,\tau(T)>}$.

5.8. END

This meaning postulate expresses that if the relation δ holds between an individual concept and a collection of properties, then there is a corresponding relation which holds between individuals. This relation is index dependent: the set of pairs which consist of a 'finder' and a 'found object', may be different for different indices. Therefore the existence of a relation between finders and found objects is formalized by means of an existential quantification over a variable which is of one intension level higher than the relation itself. An equivalent alternative would be (55), where the quantification $\exists S$ is within the scope of $\Box$ (this variant is due to P. van Emde Boas).

(55) $\Box\,[\exists S \forall x \forall P[\delta(x,P) \leftrightarrow P({}^{\wedge}\lambda y S({}^{\vee}x,{}^{\vee}y))]]$.

A notation for the relation between finder and found object is already provided by the δ_* notation. This notation is introduced in the following rule.

5.9. Reduction rule 7

Let be given an expression of the form $\delta(\alpha,\beta)$ where $\delta \in \{find, loose, eat, love, date\}$ and $\alpha \in ME_{<s,e>}$ $\beta \in ME_{\tau(T)}$. Then, replace this expression by ${}^{\vee}\beta({}^{\wedge}\lambda y[\delta_*({}^{\vee}\alpha,{}^{\vee}y)])$.

CORRECTNESS PROOF

From MP_4 follows that for all g, there is a $g' \underset{S}{\sim} g$ such that

$$g' \models \Box\,[\delta(x,P) \leftrightarrow {}^{\vee}P({}^{\wedge}\lambda y\, {}^{\vee}S({}^{\vee}x,{}^{\vee}y)).$$

This means that for all expressions $\alpha \in ME_{<s,e>}$, $\beta \in ME_{<s,\tau(T)>}$ holds that

$$g' \models \delta(\alpha,\beta) \leftrightarrow {}^{\vee}\beta({}^{\wedge}\lambda y\, {}^{\vee}S({}^{\vee}\alpha,{}^{\vee}y)).$$

For this g' the following equalities hold:

$$\begin{aligned}
&{}^{\vee}\beta({}^{\wedge}\lambda y \delta_*({}^{\vee}\alpha,{}^{\vee}y)) = \{\text{Def.5.5}\} = \beta({}^{\wedge}\lambda y \delta_*({}^{\vee}y)({}^{\vee}\alpha)) = \{\text{Def.5.7}\} = \\
&{}^{\vee}\beta({}^{\wedge}\lambda y \lambda v \lambda u \delta({}^{\wedge}u,{}^{\wedge}\lambda P[{}^{\vee}P({}^{\wedge}v)])({}^{\vee}y)({}^{\vee}\alpha)) = \{\text{choice of g'}\} = \\
&{}^{\vee}\beta({}^{\wedge}\lambda y \lambda v \lambda u[{}^{\wedge}\lambda P[{}^{\vee}P({}^{\wedge}v)]({}^{\wedge}\lambda y\, {}^{\vee}S({}^{\vee\wedge}u,{}^{\vee}y))]({}^{\vee}y)({}^{\vee}\alpha)) = \{RR_{2,1}\} = \\
&{}^{\vee}\beta({}^{\wedge}\lambda y \lambda v \lambda u[{}^{\vee\wedge}\lambda y\, {}^{\vee}S(u,{}^{\vee}y)({}^{\wedge}v)]({}^{\vee}y)({}^{\vee}\alpha)) = \{RR_{2,1}\} = \\
&{}^{\vee}\beta({}^{\wedge}\lambda y \lambda v \lambda u[{}^{\vee}S(u,{}^{\vee\wedge}v)]({}^{\vee}y)({}^{\vee}\alpha)) = \{RR_{2,1}\} = \\
&{}^{\vee}\beta({}^{\wedge}\lambda y\, {}^{\vee}S({}^{\vee}\alpha,{}^{\vee}y)) = \{\text{choice of g'}\} = \delta(\alpha,\beta).
\end{aligned}$$

Since S does not occur in the first and last formula, these expressions are equivalent for all g. From these equalities the reduction rule follows.
5.9. END

After the introduction of RR_7 we return to our discussion of (54). Application of RR_7 to (54) yields (56).

(56) ${}^{\vee\wedge}[\lambda P \exists u[unicorn_*(u) \wedge {}^{\vee}P({}^{\wedge}u)]]({}^{\wedge}\lambda y[find_*({}^{\vee\wedge}john,{}^{\vee}y)])$

This reduces further to (57), and that is the kind of formula we were looking for: it expresses that the find-relation holds between two individuals

(57) $\exists u[unicorn_*(u) \wedge find_*(john,u)]$.

The fragment contains one single verb *be*, which is used both for the be of identity, and for the copula be. An example of the be of identity is given in (58).

(58) *John is Mary.*

The first step in its production is to combine *be* with *Mary* according to S_5. This yields the IV-phrase *be Mary*. The translation of this phrase reduces by several applications of RR_1 and RR_2 to $\lambda x[{}^\vee x{=}mary]$. Combining this with *John* according to S_4 yields (58), and the corresponding translation reduces by applications of RR_1 and RR_2 to $john = mary$. One observes that the final result is an identity on the level of individuals. This shows why there is no meaning-postulate like MP_4 introduced for *be*: its translation already applies to the level of individuals rather than the level of individual concepts.

Next I give an example of the copula use of *be*.

(59) *John is a man.*

First the IV-phrase *be a man* is formed. Its translation reduces to the formula $\lambda x \exists u[man_*(u) \wedge {}^\vee x{=}u]$. Combining this with the translation of *John* yields as translation (60), which reduces to (61).

(60) $\lambda P[{}^\vee P({}^\wedge john)]\lambda x[\exists u\ man_*(u) \wedge {}^\vee x{=}u]$

(61) $\exists u[man_*(u) \wedge john{=}u]$.

In this situation one could perform one further simplification replacing (61) by (62); below I will explain why I will not do so.

(62) $man_*(john)$.

It would of course be possible to introduce a new reduction rule performing this last reduction. But it is difficult to cover the reduction from (61) to (62) by a general rule. Suppose that a rule R would say when the occurrence of a subformula $john{=}u$ implies that all occurrences of u may be replaced by *john*. In order to decide whether reduction is possible, R has to take the whole formula into consideration. Reduction from (61) to (62) is allowed, but if in (61) connective $\wedge$ would be replaced by $\rightarrow$ the reduction is not allowed. This supposed rule R would have a different character than all other reduction rules. The other rules are 'local': the question whether they may be applied, can be answered by inspecting a context of fixed length. But R would not be local because the whole formula has to be taken into account. I will not try to design such a rule R because I prefer to have only local reduction rules. Moreover, the set of reduction rules is incomplete, even with such a rule R, and only a partial solution of the reduction problem is possible. This one sees as follows. Suppose that we would define in each class of logically equivalent formulas one formula as being the simplest one (say some particular formula with shortest length). Then there exists no algorithm which reduces all formulas to the simplest in their class, since otherwise we could decide the equivalence of two formulas by reducing them to their simplest form and looking whether they are identical. Such a decision procedure would contradict the undecidability of IL (see also chapter 6, section 4).

6. EVERY MAN LOVES A WOMAN

The rules introduced in the previous sections allow us to produce sentence (63).

(63) *Every man loves a woman.*

In the introduction (section 1) I have described the two readings of this sentence. On the one reading, the same woman is loved by every man (say Brigitte Bardot), and on the other reading it might for every man be another woman (say his own mother). These two readings are represented by (64) and (65) respectively.

(64) $\exists v[woman_*(v) \wedge \forall u[man_*(u) \rightarrow love_*(u,v)]]$

(65) $\forall u[man_*(u) \rightarrow \exists v[woman_*(v) \wedge love_*(u,v)]]$.

Note that the difference between (64) and (65) is a difference in the scope of the quantifiers $\forall$ and $\exists$. Therefore this ambiguity is called a scope ambiguity. A well known variant of this scope ambiguity is (66).

(66) *Every man in this room speaks two languages.*

A remarkable aspect of the two readings of (63) is that the one reading has the other as a special case: from (64) it follows that (65) holds. Therefore one might doubt whether the two formulas really constitute an ambiguity we should deal with. One might say that the weaker formula (viz. (65)) describes the meaning of (63), and that, with additional information from the context, this can be narrowed down to the stronger one. This argument holds for (63), but I will illustrate, that it is not generally applicable. Consider (67), due to LANDMAN & MOERDIJK (1981,1983).

(67) *Every schoolboy believes that a mathematician wrote 'Through the looking glass'.*

This sentence is (at least) twofold ambiguous. On the one reading there is one mathematician of which every schoolboy believes that he wrote 'through the looking glass', but not every schoolboy necessarily believes that the person was a mathematician. On the other reading every schoolboy has the belief that some mathematician wrote the book, without necessarily having a special mathematician in mind. The rules needed for the production of sentences like (67) will be given in section 9. The formulas we will obtain then, are presented below in a somewhat simplified form. Formula (68) corresponds with the first reading (the believes concern the same mathematician), the second reading is represented by (69).

(68) $\exists v[mathematician_*(v) \wedge \forall u[schoolboy_*(u) \rightarrow believe_*(u, wrote_*(v, \textit{'Through the looking glass'}))]]$

(69) $\forall u[schoolboy_*(u) \rightarrow believe_*(u, \exists v[mathematician_*(v) \wedge wrote_*(v, \textit{'Through the looking glass'})])]$.

These two readings are logically independent: the one can be true while the other is false. The same situation arises for the well known example (66): if we read in that sentence *two* as *precisely two*, then the different scope readings are logically independent. These examples show that for variants of the scope ambiguity, both readings have to be produced by the grammar Then it is not clear why (63) should get only one reading.

A part of the production of reading (65) of sentence (63) is given in figure 8. This production is called the direct production (because no quantification rule is used).

Every man loves a woman {S, 4}
$\forall u[man_*(u) \rightarrow love(^\wedge\lambda P\exists u[woman_*(u) \wedge {}^\vee P(^\wedge u)])(^\wedge u)]$

Every man {T, 2}
$\lambda P\forall u[man_*(u) \rightarrow {}^\vee P(^\wedge u)]$

love a woman {IV, 5}
$love(^\wedge\lambda P\exists u[woman_*(u) \wedge {}^\vee P(^\wedge u)])$

Figure 8

The translation obtained in figure 8 can be reduced further by an application of RR_5, yielding (70).

(70) $\forall u[man_*(u) \rightarrow love(^\wedge u, {}^\wedge\lambda P\exists u[woman_*(u) \wedge P(^\wedge u)])]$.

Application of RR_6 yields (71), and twice application of RR_2 yields (72).

(71) $\forall u[man_*(u) \rightarrow [{}^{\vee\wedge}\lambda P\exists u[woman_*(u) \wedge {}^\vee P(^\wedge u)]](^\wedge\lambda y[love_*(^{\vee\wedge}u, {}^\vee y)])]$

(72) $\forall u[man_*(u) \rightarrow [\lambda P\exists u[woman_*(u) \wedge {}^\vee P(^\wedge u)]](^\wedge\lambda y[love_*(u, {}^\vee y)])]$.

Further application of lambda conversion is not allowed because this would bring the u in *love* $(u, {}^\vee y)$ under the scope of $\exists u$. In order to simplify this formula further, we first have to replace the variable u bound by $\exists u$ by another variable.

6.1. Reduction rule 8

Let be given an expression of the form $\lambda z\phi, \exists z\phi$ or $\forall z\phi$. Let w be a variable of the same type as z, but which does not occur in ϕ. Then replace $\lambda z\phi$, $\exists z\phi$, $\forall z\phi$ by respectively $\lambda w[w/z]\phi$, $\exists w[w/z]\phi$, and $\forall w[w/z]\phi$.

CORRECTNESS PROOF

Evident from the interpretation of these formulas.

6.1. END

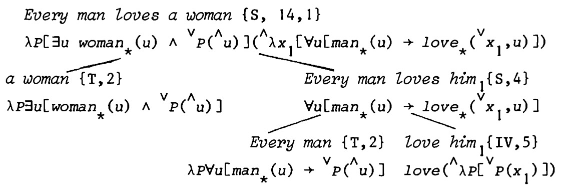
Application of RR_8 to (72) yields (73). Applications of RR_1 and RR_2 yield then (74), which reduces further to (75).

(73) $\forall u[man_*(u) \rightarrow [\lambda P\exists v[woman_*(v) \wedge {}^\vee P(^\wedge v)]](^\wedge\lambda y\ love_*(u, {}^\vee y))]$

(74) $\forall u[man_*(u) \rightarrow \exists v[woman_*(v) \wedge [\lambda y\ love_*(u, {}^\vee y)](^\wedge v)]]$

(75) $\forall u[man_*(u) \rightarrow \exists v[woman_*(v) \wedge love_*(u,v)]]$.

A part of the production of reading (64) of sentence (63) is given in figure 9. The production uses $S_{14,n}$, and it is called (for this reason) an indirect production of (63).



Every man loves a woman {S, 14,1}
$\lambda P[\exists u\ woman_*(u) \wedge {}^{\vee}P({}^{\wedge}u)]({}^{\wedge}\lambda x_1[\forall u[man_*(u) \rightarrow love_*({}^{\vee}x_1,u)]])$
a woman {T,2}
$\lambda P\exists u[woman_*(u) \wedge {}^{\vee}P({}^{\wedge}u)]$
Every man loves him$_1${S,4}
$\forall u[man_*(u) \rightarrow love_*({}^{\vee}x_1,u)]$
Every man {T,2}
$\lambda P\forall u[man_*(u) \rightarrow {}^{\vee}P({}^{\wedge}u)]$
love him$_1${IV,5}
$love({}^{\wedge}\lambda P[{}^{\vee}P(x_1)])$

Figure 9

The translation obtained in figure 9 reduces by application of RR_1 and RR_2 to (76).

(76) $\exists u[woman_*(u) \wedge \lambda x_1[\forall u[man_*(u) \rightarrow love_*({}^{\vee}x_1,u)]]({}^{\wedge}u)]$.

After change of bound variable (RR_7) we apply RR_1, and obtain (77).

(77) $\exists u[woman_*(u) \wedge \forall v[man_*(v) \rightarrow love_*(v,u)]]$.

In the introduction I have already said that sentence (78) is ambiguous; its ambiguity is called the de-dicto/de-re ambiguity. From the de-re reading (79) it follows that unicorns exist, whereas this does not follow from the de-dicto reading (80).

(78) *John seeks a unicorn*

(79) $\exists u[unicorn_*(u) \wedge seek_*(john,u)]$

(80) $seek({}^{\wedge}john, {}^{\wedge}\lambda P\exists u[unicorn_*(u) \wedge P({}^{\wedge}u)])$.

This ambiguity can be considered as a scope ambiguity: the difference between (79) and (80) is the difference in scope of the existential quantifier. Note that formulas (79) and (80) are logically independent, hence we have to produce both readings.These productions are analogous to the productions of the different scope readings of *Every man loves a woman*. The de-dicto reading (80) is obtained by a direct production. We have considered this production in the previous section. The de-re reading, viz. (79), is obtained by an indirect production. As a first stage of the indirect production sentence (81) is formed, which has (82) as translation.

(81) *John seeks him$_1$*

(82) $seek({}^{\wedge}john, {}^{\wedge}\lambda P[{}^{\vee}P(x_1)])$.

Combination according to $S_{14,1}$, of (81) with the term *a unicorn* yields (78), and combination of their translations according to $T_{14,1}$ yields (83), reducing to (84).

(83) $\lambda P \exists u[unicorn_*(u) \wedge {}^{\vee}P({}^{\wedge}u)]({}^{\wedge}\lambda x_1[{}^{\wedge}seek({}^{\wedge}john,\ {}^{\wedge}\lambda P[{}^{\vee}P(x_1)])])]$

(84) $\exists u[unicorn_*(u) \wedge seek({}^{\wedge}john,\ {}^{\wedge}\lambda P[{}^{\vee}P({}^{\wedge}u)])]$.

Application of RR_6 reduces this formula to (85).

(85) $\exists u[unicorn_*(u) \wedge seek_*(john,u)]$.

Sentence (86) can be produced using the same syntactic rules as in the production of (78).

(86) *Mary finds a unicorn.*

This sentence is not ambiguous; it only has a referential reading. In the previous section it was explained how the translation of the direct production reduces to such a reading. The indirect production yields, of course, a referential reading as well. An interesting aspect of the indirect production is the way in which the obtained formulas can be reduced. For this reason I will consider this production in more detail. A first stage of the indirect production of (86) is (87), which has (88) as translation.

(87) *Mary finds him$_1$.*

(88) $find({}^{\wedge}mary,\ {}^{\wedge}\lambda P[{}^{\vee}P(x_1)])$.

One method to reduce (88) is to apply the same reduction rules as used in the reduction of (82). Then as last step RR_6 is applied, see the reduction of (84). But another reduction process is possible as well. We might apply RR_7 to (88) because meaning postulate 4 holds for *find*. Thus we obtain (89), reducing to (90).

(89) ${}^{\wedge}\lambda P[{}^{\vee}P(x_1)]({}^{\wedge}\lambda y[find_*(mary,\ {}^{\vee}y)])$

(90) $find_*(mary,\ {}^{\vee}x_1)$.

Combination, according to $S_{14,1}$ of (90) with the translation of *a unicorn* yields (91), which reduces to (92).

(91) $\lambda P \exists u[unicorn_*(u) \wedge {}^{\vee}P({}^{\wedge}u)]({}^{\wedge}\lambda x_1[find_*(mary,{}^{\vee}x_1)])$

(92) $\exists u[unicorn_*(u) \wedge find_*(mary,u)]$.

This shows that there are two methods to reduce the formulas obtained in the indirect production of (86).

In general it makes no difference in which order we apply the reduction rules. Sooner or later we have to apply the same rule to the same (sub)expression. An exception is the introduction of δ_* for constants to which meaning postulate 4 applies. Once we have applied the meaning postulate (i.e. RR_7), we cannot apply the definition for δ_* (i.e. RR_6) any more. The reason for this is that both applications consume an occurrence of δ, and produce an occurrence of δ_*. As practice learns, these two ways of reduction always yield the same result. I have, however, not a formal proof of some formal version of this observation. The situation is difficult due to the interaction of RR_6 and RR_7 with many other reduction rules. In FRIEDMAN & WARREN (1979) related reduction rules are considered, and they provide several examples of the complex interactions of the rules (they have no normal form theorem for their system either).

Finally I consider a sentence which is not ambiguous. For sentence (93) the de-re reading is the only possible reading, and it is the only reading produced by the grammar.

(93) *John seeks a unicorn and Mary seeks it.*

The occurrence of *it* requires an application of $S_{14,n}$. A part of the production of (93) is given in figure 10.

John seeks a unicorn and Mary seeks it
$\exists u[unicorn_*(u) \wedge \lambda x_1[seek(^\wedge john, {}^\wedge\lambda P[^\vee P(x_1)]) \wedge seek(^\wedge mary, {}^\wedge\lambda P[^\vee P(x_1)])]u]$

a unicorn
$\lambda P[\exists u[unicorn_*(u) \wedge {}^\vee P(^\wedge u)]$

John seeks him$_1$ and Mary seeks him$_1$
$seek(^\wedge john, {}^\wedge\lambda P^\vee P(x_1)) \wedge seek(^\wedge mary, {}^\wedge\lambda P^\vee P(x_1))$

John seeks him$_1$
$seek(^\wedge john, {}^\wedge\lambda P[^\vee P(x_1)])$

Mary seeks him$_1$
$seek(^\wedge mary, {}^\wedge\lambda P[^\vee P(x_1)])$

Figure 10

The obtained translation for (93) reduces to (94).

(94) $\exists u[unicorn_*(u) \wedge seek_*(john,u) \wedge seek_*(mary,u)]$.

7. BILL WALKS IN THE GARDEN

In this section the fragment is extended with the categories Prep of prepositions, and IAV of IV-modifying adverbials. In PTQ the category 'IAV' is also called 'IV/IV'. For the basic elements of IAV there are corresponding constants of type $<<s,\tau(IV)>,\tau(IV)>$. The definitions concerning IAV are as follows.

7.1. B_{IAV} = {*slowly,voluntarily,allegedly*}

7.2. {*slowly,voluntarily,allegedly*} $\subset$ $CON_{\tau(IAV)}$

7.3. *slowly'* = *slowly*, *voluntarily'* = *voluntarily*, *allegedly'* = *allegedly*.

7.3. END

An adverb forms with an IV-phrase, according to S_{10}, a new IV-phrase.

7.4. Rule S_{10}:

IAV × IV → IV

F_{10}: concatenate (α,β)
T_{10}: $\alpha'(^{\wedge}\beta')$.

7.4. END

An example of a sentence containing an IAV is (95).

(95) *John voluntarily walks.*

The production of (95) is presented in figure 11.

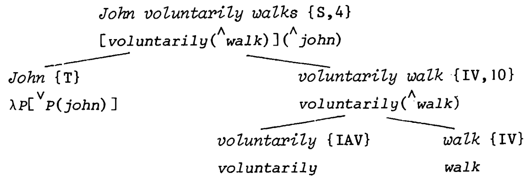


Figure 11

In PTQ the convention was introduced to write all expressions of the form $\gamma(\alpha)(\beta)$ as $\gamma(\beta,\alpha)$. This example shows that the PTQ formulation was too liberal: it would allow to write *voluntarily* as a relation: *voluntarily*($^\wedge$*john*, $^\wedge$*walk*). This result is not attractive because traditionally one does not consider *voluntarily* as a relation. Therefore in reduction rule 5 this convention was only introduced for γ being a verb.

The translation obtained for (95) does not allow for the conclusion that John walks, although this would be a correct conclusion from sentence (95). Not all adverbs allow for such a conclusion. From (96) it does not follow that John walks.

(96) *John allegedly walks.*

This means that the adverb *allegedly* creates an intensional context for the object of a verb. Also sentence (97) does not allow to conclude to the existence of a unicorn.

(97) *John allegedly loves a unicorn.*

One might expect the introduction of a meaning postulate that expresses the extensional character of *slowly* and *voluntarily*. Such a meaning postulate is not given in PTQ. I expect that it would be of a different nature than the postulates we have met before: it would be an implication, and I expect that it would not give rise to simplifications of the formulas involved.

The fragment contains two prepositions, and from these new adverbial-phrases can be formed. Prepositions translate into constants of type $<<s,\tau(T)>,\tau(IAV)>$.

7.5. $B_{Prep} = \{in, about\}$

7.6. $\{in, about\} \subset CON_{\tau(Prep)}$

7.7. $in' = in$, $about' = about$.

7.7. END

The rule for creating new adverbs is as follows.

7.8. Rule S_6:

Prep $\times$ T $\rightarrow$ IAV

F_6: concatenate (α,β)

T_6: $\alpha'(^\wedge\beta')$.

7.8. END

An example of an application of this rule is given in figure 12, where sentence (98) is produced.

(98) *John talks about a unicorn.*

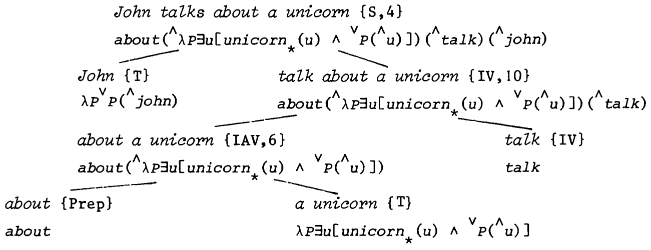

John talks about a unicorn {S,4}
$about(^\wedge\lambda P\exists u[unicorn_*(u) \wedge {}^\vee P(^\wedge u)])(^\wedge talk)(^\wedge john)$
John {T}
$\lambda P\,{}^\vee P(^\wedge john)$
talk about a unicorn {IV,10}
$about(^\wedge\lambda P\exists u[unicorn_*(u) \wedge {}^\vee P(^\wedge u)])(^\wedge talk)$
about a unicorn {IAV,6}
$about(^\wedge\lambda P\exists u[unicorn_*(u) \wedge {}^\vee P(^\wedge u)])$
talk {IV}
talk
about {Prep}
about
a unicorn {T}
$\lambda P\exists u[unicorn_*(u) \wedge {}^\vee P(^\wedge u)]$

Figure 12

The translation obtained here does not imply that there is a unicorn John talks about: *about* creates an intensional context. This is the result we aimed at (see section 1).

In the same way as we produced (98), we may produce (99) with as translation (100).

(99) *Bill walks in the park*

(100) $in(^\wedge\lambda P\exists u[\forall v[park_*(v) \leftrightarrow u=v] \wedge {}^\vee P(^\wedge u)])(^\wedge walk)(^\wedge bill)$.

This result is not completely satisfactory. If Bill walks in the park, then one may conclude that there exists a park, and if the park is the Botanical garden, then from (99) it may be concluded that Bill walks in the Botanical garden. So the locative preposition *in* does not create an intensional context. This property of *in* is formalized in the following meaning postulate.

7.9. Meaning postulate 8

$$\exists G\forall P\forall Q\forall x\Box\,[in(P)(Q)(x) \leftrightarrow {}^\vee P(^\wedge\lambda y[[{}^\vee G]({}^\vee y)(Q)(x)])]$$

7.9. END

In order to be able to give a reduction rule on the basis of this meaning postulate, a notation for the predicate denoted by ${}^\vee G$ in MP8 is introduced (such a notation for prepositions is not defined in PTQ). This notation is chosen in analogy of the notation δ_* for verbs.

7.10. DEFINITION.

$$\delta_* = \lambda x \lambda Q \lambda u[\delta(^\wedge\lambda P[^\vee P(^\wedge u)])(Q)(x))] \quad \text{where} \quad \delta \in CON_{\tau(Prep)}.$$

7.10. END

On the basis of this definition we have the following reduction rule.

7.11. Reduction rule 9

Let be given an expression of the form $in(\alpha)(\beta)(\gamma)$, where $\alpha \in ME_{<s,\tau(T)>}$, $\beta \in ME_{<s,\tau(IV)>}$, $\gamma \in ME_{<s,e>}$. Then replace this expression by $^\vee\alpha(^\wedge\lambda y[in_*(^\vee y)(\beta)(\gamma)])$.

CORRECTNESS PROOF. Let $v \in VAR_e$, $x \in VAR_{<s,e>}$ and $Q \in VAR_{<s,\tau(IV)>}$. Then for all g

$$g \models in_*(v)(Q)(x) = in(^\wedge\lambda P\, ^\vee P(^\wedge v))(Q)(x).$$

We now apply MP8 to the right hand side of the equality: this meaning postulate says that there is a $g' \underset{G}{\sim} g$ such that

$$g' \models in_*(v)(Q)(x) = [^{\vee\wedge}\lambda P\, ^\vee P(^\wedge v)](^\wedge\lambda y[[^\vee G](^\vee y)(Q)(x)]).$$

The expression to the right of the equality sign reduces by means of several applications of RR_1 and RR_2. Thus we obtain

$$g' \models in_*(v)(Q)(x) = [^\vee G](v)(Q)(x).$$

Consequently $g' \models in_* = {}^\vee G$. This means that from MP_8 it follows that

$$\models \forall P \forall Q \forall x \Box\, [in(P)(Q)(x) \leftrightarrow {}^\vee P(^\wedge\lambda y[in_*(^\vee y)(Q)(x)])].$$

7.11. END

Formula (100) can be reduced, using RR_9, to (101) and further to (102)

(101) $[^{\vee\wedge}\lambda P[\exists u \forall v[park_*(u) \leftrightarrow u=v] \wedge {}^\vee P(^\wedge u)]](^\wedge\lambda y[in_*(^\vee y)(^\wedge walk)(^\wedge bill)])$

(102) $\exists u \forall v[park_*(v) \leftrightarrow u=v] \wedge in_*(u)(^\wedge walk)(^\wedge bill)]$.

In PTQ no examples concerning the meaning postulate for *in* are given. This example illustrates the consequence of the meaning postulate: if one stands in the relation of walking in with 'a collection of properties', then there is an 'individual' with which one has this relation.

8. JOHN TRIES TO FIND A UNICORN

In this section a new category of IV-modifiers is introduced. This new category is called IV//IV (IV modifying verbs) and contains verbs taking verbs as complements. The fragment has only two of such verbs (*try to, wish to*), although there are a lot more in English. The syntactic treatment of these verbs is rather primitive: *try to* is considered as a single word containing a space (so *to* is not treated as a word). But our main interest is semantics, and the verbs are interesting in this respect. They create intensional contexts even when the sentence without such a verb would only have a de-re reading. An example is (103); this sentence does not necessarily have the implication that unicorns exist.

(103) *John tries to find a unicorn.*

Corresponding with the verbs of category IV//IV there are constants in the logic of the type $<<s,\tau(IV)>,\tau(IV)>$. The verbs translate into these constants.

8.1. $B_{IV//IV} = \{try\ to,\ wish\ to\}$

8.2. $\{try\ to,\ wish\ to\} \subset CON_{\tau(IV//IV)}$

8.3. *try to'* = *try to*, *wish to'* = *wish to*.

8.3. END

The members of IV//IV are used in the following rule.

8.4. Rule S_8:

$IV//IV \times IV \rightarrow IV$

F_8: concatenate (α,β)

T_8: $\alpha'(^\wedge\beta')$.

8.4. END

The production of (103) is partially presented in figure 13.

John tries to find a unicorn {S,4}
$try\ to(^{\wedge}john,\ ^{\wedge}find(^{\wedge}\lambda P\,^{\wedge}\exists u[unicorn_{*}(u) \wedge\ ^{\vee}P(^{\wedge}u)]))$

John
$\lambda P\,^{\vee}P(^{\wedge}john)$

try to find a unicorn {IV,8}
$try\ to(^{\wedge}find(^{\wedge}\lambda P\exists u[unicorn_{*}(u) \wedge\ ^{\vee}P(^{\wedge}u)]))$

try to
$try\ to$

find a unicorn {IV,5}
$find(^{\wedge}\lambda P\exists u[unicorn_{*}(u) \wedge\ ^{\vee}P(^{\wedge}u)])$

Figure 13

The formula obtained in this production process does not reduce further, and it does not allow to conclude for the existence of a unicorn which John tries to find. So the de-dicto aspect is dealt with adequately. But sentence (103) can also be used in a situation in which there is a unicorn which John tries to find. For reasons related to the ones given concerning *John seeks a unicorn,* the reading involving a particular unicorn has to be obtained as an alternative translation for (103). That reading can be obtained using $S_{14,n}$.

The translation obtained for (103) in figure 13 is, however, not in all respects satisfactory. We do not get information concerning the relation between John and the property expressed in the second argument of *try to*. In particular it is not expressed that what John tries to achieve is that John (he himself) finds the unicorn, and not that someone else finds the unicorn. For verbs like *promise* and *permit* the relation between the subject and the complement is much more complex. A correction of this disadvantage of the PTQ treatment can be found along the lines of DOWTY (1978) and BARTSCH (1978b), see also section 4.1 in chapter VII.

In section 4 we introduced the rules for IV conjunction and disjunction. The verb phrases involved may concern two coreferential terms as in (104).

(104) *John finds a unicorn and eats it.*

The coreferentiality can be dealt with by means of quantifying in the term *a unicorn*. This yields the reading (105).

(105) $\exists u[unicorn_{*}(u) \wedge find_{*}(john,u) \wedge eat_{*}(john,u)]$.

This formula expresses that there is a particular unicorn which John finds and eats.

The conjoined verb phrase underlying (106) can be embedded in a *try to* construction.

(106) *John tries to find a unicorn and eat it.*

This sentence does not allow for the conclusion that there is a unicorn. The occurrence of a pronoun, however, invites us to produce this sentence with quantification rule S_{14}, and that would result in a referential reading, viz. (107)

(107) $\exists u[\mathit{unicorn}_*(u) \wedge \mathit{try\ to}(^\wedge \mathit{john}, ^\wedge[\mathit{find}(\lambda P[^\vee P(^\wedge u)]) \wedge \mathit{eat}(\lambda P[^\vee P(^\wedge u)])])]$.

A new quantification rule makes it possible to produce (106) in a reading which does not imply the existence of a unicorn. The following rule scheme describes the quantification of a Term into an IV-phrase.

8.5. <u>Rule $S_{16,n}$:</u>

$T \times IV \rightarrow IV$

$F_{16,n}$: If α does not have the form he_k
then replace in β the first occurrence of he_n or him_n
by α, and all other occurrences of he_n by *he/she/it* and of
him_n by *him/her/it* according to the gender of the first T
or CN in α
else replace all occurrences of he_n by he_k and of him_n by him_k.

$T_{16,n}$: $\lambda y[\alpha'(^\wedge \lambda x_n[\beta'(y)])]$.

8.5. END

In order to produce (106) we first produce the verbphrase (108).

(108) *find a unicorn and eat it.*

The production of (108) is partially given in figure 14.

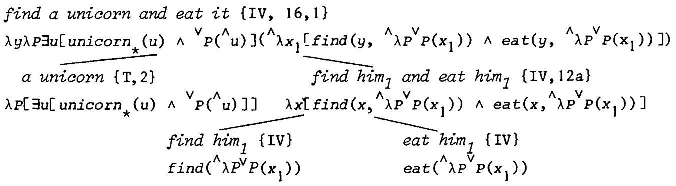

find a unicorn and eat it {IV, 16,1}
$\lambda y \lambda P \exists u[unicorn_*(u) \wedge {}^{\vee}P({}^{\wedge}u)]({}^{\wedge}\lambda x_1[find(y, {}^{\wedge}\lambda P\, {}^{\vee}P(x_1)) \wedge eat(y, {}^{\wedge}\lambda P\, {}^{\vee}P(x_1))])$
a unicorn {T,2}
$\lambda P[\exists u[unicorn_*(u) \wedge {}^{\vee}P({}^{\wedge}u)]]$
find him₁ and eat him₁ {IV,12a}
$\lambda x[find(x, {}^{\wedge}\lambda P\, {}^{\vee}P(x_1)) \wedge eat(x, {}^{\wedge}\lambda P\, {}^{\vee}P(x_1))]$
find him₁ {IV}
$find({}^{\wedge}\lambda P\, {}^{\vee}P(x_1))$
eat him₁ {IV}
$eat({}^{\wedge}\lambda P\, {}^{\vee}P(x_1))$

Figure 14

Now we return to the production of sentence (106). Its production from (108) is presented in figure 15.

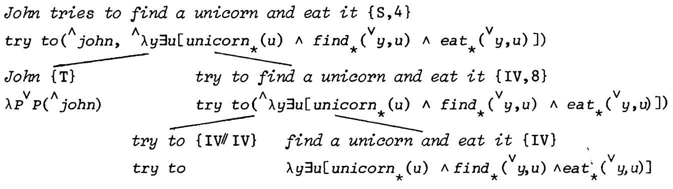

John tries to find a unicorn and eat it {S,4}
$try\ to({}^{\wedge}john, {}^{\wedge}\lambda y \exists u[unicorn_*(u) \wedge find_*({}^{\vee}y,u) \wedge eat_*({}^{\vee}y,u)])$
John {T}
$\lambda P\, {}^{\vee}P({}^{\wedge}john)$
try to find a unicorn and eat it {IV,8}
$try\ to({}^{\wedge}\lambda y \exists u[unicorn_*(u) \wedge find_*({}^{\vee}y,u) \wedge eat_*({}^{\vee}y,u)])$
try to {IV//IV}
$try\ to$
find a unicorn and eat it {IV}
$\lambda y \exists u[unicorn_*(u) \wedge find_*({}^{\vee}y,u) \wedge eat_*({}^{\vee}y,u)]$

Figure 15

A sentence related with (106) is (109).

(109) *John tries to find a unicorn and wishes to eat it.*

Montague argues that only a referential reading of this sentence is possible (except for the case that the pronoun *it* is considered as a pronoun of laziness). A production of sentence (109) might be given in which S_{14} is used. Then it is not surprising that a referential reading is obtained. But this is also the case for a production using S_{16}, as will be shown below. The first step is to form verb phrase (110), with translation (111).

(110) *try to find him₁ and wish to eat him₁*

(111) $\lambda x[try\ to(x, {}^{\wedge}find({}^{\wedge}\lambda P\, {}^{\vee}P(x_1))) \wedge wish\ to(x, {}^{\wedge}eat({}^{\wedge}\lambda P\, {}^{\vee}P(x_1)))]$.

Combination of (110) with *a unicorn* according to $S_{16,1}$ yields (112). The translation is (113), which reduces to (114).

(112) *try to find a unicorn and wish to eat it*

(113) $\lambda y[\lambda P \exists u[unicorn_*(u) \wedge {}^{\vee}P({}^{\wedge}u)]({}^{\wedge}\lambda x_1[try\text{-}to(y, {}^{\wedge}find({}^{\wedge}\lambda P\, {}^{\vee}P(x_1))) \wedge$
$wish\ to(y, {}^{\wedge}eat({}^{\wedge}\lambda P\, {}^{\vee}P(x_1)))])]$

(114) $\lambda y \exists u[unicorn_*(u) \wedge try\ to(y, {}^{\wedge}find({}^{\wedge}\lambda P\, {}^{\vee}P({}^{\wedge}u))) \wedge$
$wish\ to(y, {}^{\wedge}eat({}^{\wedge}\lambda P\, {}^{\vee}P({}^{\wedge}u)))]$.

Combination of (112) with *John* according to S_4 yields sentence (109). The translation is (115).

(115) $\exists u[unicorn_*(u) \wedge try\text{-}to({}^{\wedge}john, {}^{\wedge}find({}^{\wedge}\lambda P\, {}^{\vee}P({}^{\wedge}u))) \wedge$
$wish\text{-}to({}^{\wedge}john, {}^{\wedge}eat({}^{\wedge}\lambda P\, {}^{\vee}P({}^{\wedge}u)))]$.

The formula obtained here can be simplified by replacing $\delta({}^{\wedge}\lambda P\, {}^{\vee}P({}^{\wedge}u))$ by $\lambda y\ \delta({}^{\wedge}\lambda P\, {}^{\vee}P({}^{\wedge}u))(y)$, where δ is the translation of a transitive verb. The advantage of this replacement is that now RR_5 and RR_6 can be used. In this way (115) reduces to (116).

(116) $\exists u[unicorn_*(u) \wedge try\text{-}to({}^{\wedge}john, {}^{\wedge}\lambda y\ find_*({}^{\vee}y,u)) \wedge$
$wish\ to({}^{\wedge}john, {}^{\wedge}\lambda y\ eat_*({}^{\vee}y,u))]$

This method is formulated in a reduction rule as follows.

8.6. Reduction rule 10

Let be given an expression of the form $\delta({}^{\wedge}\lambda P\, {}^{\vee}P({}^{\wedge}u))$, where δ is the translation of a TV for which MP_4 holds. Then replace this expression by $\lambda y \delta_*({}^{\vee}y,u)$.

CORRECTNESS PROOF. By definition of interpretation the two expressions are equivalent.
8.6. END

The possibilities for application of RR_{10} are limited by mentioning explicitly the argument of δ. One might omit this argument; then the rule would be applicable in many more circumstances, for instance to the formula obtained in figure 13. I have not used this more general version because it would not give rise to simpler formulas (in the sense of more concise formulas), but one might judge that the general rule would give rise to more understandable formulas.

9. JOHN BELIEVES THAT MARY WILL RUN

A new construction considered in this section arises from verbs of the category IV/S; I.e. verbs taking a sentence as complement. There are several such verbs, but only two of them are incorporated in the fragment.

9.1. $B_{IV/S}$ = {*believe that, assert that*}

9.2. {*believe that, assert that*} $\subset CON_{<<s,\tau(S)>,\tau(IV)>>}$

9.3. *believe that'* = *believe that, assert that'* = *assert that.*

9.3. END

The rule producing IV phrases from these verbs reads as follows.

9.4. Rule S_7:

IV/S × S → IV

F_7: concatenate (α,β)
T_7: $\alpha'(^\wedge\beta')$.

9.4. END

An example of a sentence with a verb of category IV/S is (117).

(117) *John believes that Mary runs.*

Part of the production of (117) is given in figure 16.

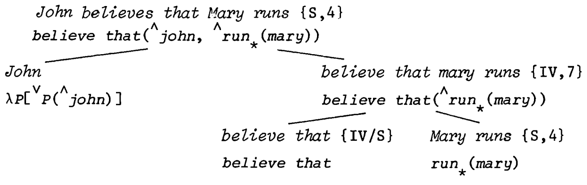

John believes that Mary runs {S,4}
believe that(^john, ^run*(mary))
John
λP[ᵛP(^john)]
believe that mary runs {IV,7}
believe that(^run*(mary))
believe that {IV/S}
believe that
Mary runs {S,4}
run*(mary)

Figure 16

Believe is considered as a relation between an individual concept and a proposition (i.e. a function from indices to truth values). It is not said what kind of relation this is. There are several proposals in the literature analyzing the believe relation in more detail (e.g. LEWIS 1970), but Montague did not analyze it any further.

The formula obtained in figure 16 expresses that *believe* is a relation with as first argument $^{\wedge}john$. To this, the same comment applies as to the first argument of the seek-relation: there is no generally accepted notation which expresses that for this argument *believe* can be considered as a relation with as first argument an individual. The second argument is an expression of type <s,t>. Would it have been an expression of type t, then we could replace it by any other expression which denotes (for the current index) the same truth value. So if someone would believe a truth, he would believe all truths (for the current index). Now that the second argument of the believe-relation is a proposition, this is not the case. If John and Mary walk, then one may believe that John walks, without having the formal implication that one believes that Mary walks. Nevertheless, the use of a proposition is not completely satisfactory. It implies that in case John believes a tautology, he believes all tautologies. This is a fundamental shortcoming of this kind of approach; there is, however, not a generally accepted alternative.

The aspect that makes the introduction of *believe* and *assert* interesting in the present fragment, even with the present semantics, is that these verbs introduce intensional contexts in which a de-re reading is impossible. Sentence (118) does not allow for the conclusion that there exists a unicorn.

(118) *Mary believes that John finds a unicorn and he eats it.*

The relevant part of the production of sentence (120) is given in figure 17.

Mary believes that John finds a unicorn and he eats it
$believe\ that(^{\wedge}mary,\ ^{\wedge}\exists u[unicorn_*(u) \wedge find_*(john,u) \wedge eat_*(john,u)])$

Mary
$\lambda P[^{\vee}P(^{\wedge}mary)]$

believe that John finds a unicorn and he eats it
$believe\ that(^{\wedge}\exists u[unicorn_*(u) \wedge find_*(john,u) \wedge eat_*(john,u)])$

Figure 17

A further extension of the fragment are the restrictive relative clauses: terms will be produced like *Every man such that he runs*. This *such that* form is not the standard form of relative clauses, but it avoids the syntactic complications arising from the use of relative pronouns. The following rule scheme describes how relative clause constructions are formed out of a CN and a sentence.

9.5. <u>Rule $S_{3,n}$:</u>

$CN \times S \rightarrow CN$

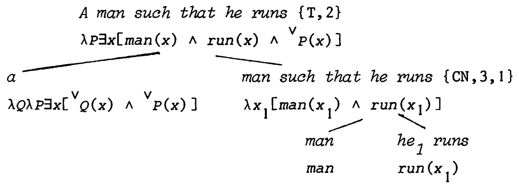
$F_{3,n}$: replace in β all occurrences of he_n by *he/she/it* and him_n by *him/her/it* according to the gender of the first CN in α.

$T_{3,n}$: $\lambda x_n[\alpha'(x_n) \wedge \beta']$.

9.5. END

An example is the production of term (119), which is given in figure 18.

(119) *a man such that he runs.*


A man such that he runs {T,2}
$\lambda P\exists x[man(x) \wedge run(x) \wedge {}^{\vee}P(x)]$
a
$\lambda Q\lambda P\exists x[{}^{\vee}Q(x) \wedge {}^{\vee}P(x)]$
man such that he runs {CN,3,1}
$\lambda x_1[man(x_1) \wedge run(x_1)]$
man
man
he_1 *runs*
$run(x_1)$

<u>Figure 18</u>

The obtained translation can be reduced, using RR_4, to (120).

(120) $\lambda P\exists u[man_*(u) \wedge run_*(u) \wedge {}^{\vee}P({}^{\wedge}u)]$.

Rule $S_{3,n}$ takes a CN as one of its arguments, and yields a CN as result. This means that the rule can be applied more than one time in succession. Then terms are obtained like the one in (121)

(121) *Every man such that he walks such that he talks.*

In (121) both the relative clauses are attached to the head *man*; this phenomenon is called 'stacking'. A situation that may arise in connection with stacking is as follows. The second relative clause contains a pronoun which is coreferential with a term in the first relative clause, whereas the pronoun is (semantically) within the scope of the determiner of the whole term. An example, due to Bresnan (PARTEE 1975, p.263) is (124).

(122) *Every girl who attended a womans college who gave a donation to it, was put on the list.*

Sentence (124) exhibits co-reference within the compound CN phrase: *it* in the second relative clause refers to the college in the first relative clause. The whole term has a reading in which the college needs not to be the same for all girls. Suppose that we obtained coreferentiality by means of quantifying in the term *a womans college* for *him*$_1$ in sentence (123).

(123) *Every girl who attended him*$_1$ *who gave a donation to him*$_1$ *was put on the list.*

In that production process a reading would be obtained with for the existential quantifier wider scope than for the universal quantifier. That is not the intended reading. In order to obtain the intended reading, a new quantification rule is introduced: quantification into a CN phrase.

9.6. Rule $S_{15,n}$:

$T \times CN \rightarrow CN$

$F_{15,n}$: Replace the first occurrence of *he*$_n$/*him*$_n$ in β by α.
Replace all other occurrences of *he*$_n$ by *he/she/it*, and of *him*$_n$ by *him/her/it*, according to the gender of the first CN or T in α.

$T_{15,n}$: $\lambda y \alpha'(^\wedge\lambda x_n[\beta'(y)])$.

9.6. END

An extensive discussion of relative clause formation will be given in chapter 8; examples in which rule S_{15} is used, will be considered in appendix 1. There also will be solved a problem that I neglected above: reduction rule RR_4 applies to the translation of terms like (120), but not to such terms with the determiners *every* or *the*.

In the remainder of this section I mention some rules which are introduced only to incorporate the complete PTQ fragment. The first rule concerns the sentence modifier *necessarily*.

9.7. $B_{S/S}$ = *necessarily*

9.8. *necessarily'* = $\lambda p \Box [^\vee p]$.

9.9. <u>Rule S_9:</u>

$S/S \times S \rightarrow S$

F_9: concatenate (α,β)
T_9: $\alpha'(^\wedge\beta')$.

9.9. END

An example is the production of (124) which gets as its translation (127).

(124) *Necessarily John runs.*

(125) $\Box\ run_*(john)$.

This example illustrates how sentence modifiers can be incorporated in the fragment. The translation of (126) is not correct since that sentence cannot mean that John always runs. For an alternative of the semantics of *necessarily* see e.g. VELTMAN 1980.

Up till now we have met sentences in the positive present tense. PTQ has rules for some other tenses as well. These rules have several shortcomings, and I will mention them without further discussion.

9.10 <u>Rule S_{17a}:</u>

$T \times IV \rightarrow S$

F_{17a}: replace the first verb in β by its negative third person singular present; concatenate(α,β)
T_{17a}: $\neg\alpha'(^\wedge\beta')$

9.11 <u>Rule S_{17b}:</u>

$T \times IV \rightarrow S$

F_{17b}: replace the first verb in β by its third person singular future; concatenate(α,β)
T_{17b}: $W\alpha'(^\wedge\beta')$

9.12 <u>Rule S_{17c}:</u>

$T \times IV \rightarrow S$

F_{17c}: replace the first verb in β by its negative third person singular future; concatenate(α,β)
T_{17c}: $\neg W[\alpha'(^\wedge\beta')]$

9.13 <u>Rule S_{17d}:</u>

$T \times IV \rightarrow S$

F_{17d}: replace the first verb in β by its third person singular perfect. concatenate(α,β)

T_{17d}: $H[\alpha'(^{\wedge}\beta')]$

9.14 <u>Rule S_{17e}:</u>

$T \times IV \rightarrow S$

F_{17e}: replace the first verb in β by its negative person singular present perfect; concatenate(α,β)

T_{17e}: $\neg H[\alpha'(^{\wedge}\beta')]$.

9.14. END

This completes the exposition of the PTQ fragment. One should realize that the sentences we have discussed, constitute a special selection of the sentences of the fragment. Besides those rather natural examples, there are a lot of remarkable sentences in the fragment. An example is (128).

(126) *The park walks in John.*

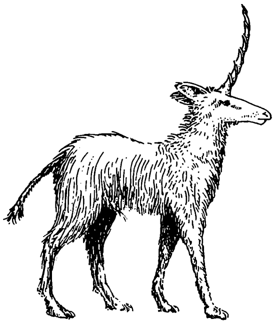
Whether this is a shortcoming or not, depends on the opinion one has about the acceptability of (126). And how this should be dealt with, depends on the opinion one has about the question which component of the grammar should deal with such phenomena. Since these questions are completely independent of the problems we were interested in, I have not discussed this aspect. Several more fundamental aspects of the system which were not completely satisfactory, have been mentioned in the discussions. Other such aspects will arise in the discussion in later chapters, for instance in appendix . As for the main aim of the enterprise, I conclude that Montague has for the problematic sentences mentioned in section 1 indeed provided an analysis which has the desired semantic properties, and which is in accordance with the compositional framework.

CHAPTER VI

VARIANTS AND DEVIATIONS

ABSTRACT

In this chapter the impact of the algebraic framework on the design of grammars, is illustrated by considering several proposals from the literature. Most of these proposals contain details which are not in accordance with the framework. It will be shown that these proposals can be improved by adopting an approach which is in accordance with the framework, without losing the semantic effect the proposal was designed for. Other proposals present acceptable variants for certain details of the framework.



1. INTRODUCTION

On the basis of several proposals from the literature, I will illustrate in this chapter what the practical consequences are of the framework we have developed in chapters 1 and 2. Some of the proposals were already discussed in JANSSEN 1978a. The rules from the proposals will not be adapted to the way of presentation used up till now, but they are quoted in the way they were formulated in the original papers. I expect that this will cause no problems. Only the formulas of IL are sometimes adapted to our notations (e.g. $^\wedge\lambda$ instead of $\hat{\lambda}$). Some of the proposals concern variants which are in accordance with the framework, but most are not. The objections against these proposals, however, concern in most cases only a minor detail of the paper, and my criticism should not be taken as a criticism on the paper as a whole. On the contrary, most of the papers I like very much, and that was a reason for studying them in detail. I will not consider proposals which are presented in such a way that it is evident that they are intended as a non-compositional component of the system (e.g. the indexing component for variables of COOPER & PARSONS 1976, and the interpretation strategy for pronouns of BARTSCH 1979). Rather I will discuss aspects of proposals which seem at first glance to be in accordance with the framework, but which at closer investigation appear not to be. Such examples exhibit that the practical consequences of the framework are sometimes not well understood. These examples are collected here to provide as illustrations of the framework: non-examples too can be very instructive. I hope that the examples give the reader an improved understanding of what it means to design a Montague grammar. As a matter of fact, my personal experience with the examples discussed here, was a great stimulans for the research presented in this book: discovering the foundations of Montague grammar, and investigating the practical consequences of these fundamental properties.

The structure of our framework, as developed in chapters 1 and 2, is presented in figure 1. The arrows 2,5, and 7, are homomorphisms, and the arrows 3 and 6 are derivers. The examples we will consider are grouped according to the arrow representing the component where the deviation from the framework can be located. The number of the arrow indicates the section where that group of examples will be considered.

$$\begin{tikzcd}
 & T_E \arrow[d, "2"] \\
IL \arrow[r, dashed, "3"] \arrow[d, "5"'] & IL' \arrow[d, "7"] \\
M \arrow[r, dashed, "6"'] & M'
\end{tikzcd}$$

T_E	: Term algebra of grammar for English
IL	: Intensional Logic
IL'	: Translation of English into algebra derived from intensional logic
M	: Meanings for intensional logic
M'	: Meanings for English

Figure 1. The framework

The framework of Montague grammar constitutes a framework which guarantees that one is working in accordance with the principle of compositionality. Deviations from this framework are not just deviations from some arbitrary mathematical system, but from a framework that is designed with the purpose of both obeying the principle, and being at the same time as general as possible. If one violates this framework, then there is a great risk that one does not only disturb the framework, but also the underlying principle of compositionality. The ultimate consequence may be that one does not describe a semantics at all. In the discussion it will turn out that the practical examples of violations of the framework in most cases indeed yield an incorrect (i.e. unintended) semantics, or no semantics at all. In such cases the framework guides us toward a correct solution. In other cases, where the proposal did not give rise to an incorrect semantics, the principle suggests another kind of solution that is simpler than the original proposal. These aspects exhibit the value of (the formalization of) the principle of compositionality as a heuristic tool.

In the light of the above remarks, it is useful to give a characterization of what are harmful deviations of Montague's framework, and what are harmless variants. This characterization can be given at the hand of figure 1. It is harmless to change the language of which the semantics is given; to change the kind of logic used as auxiliary language, or to change the kind of meanings obtained. All algebras in the figure may be replaced by other algebras. But the algebraic relations between them should not be changed; the algebraic properties of the arrows should not be disturbed. Homomorphisms should remain homomorphisms, and derivers should remain derivers. These are the properties which guarantee that the principle of compositionality is obeyed.

2. THE USE OF SYNTACTIC INFORMATION

2.1. Introduction

Some proposals from the literature contain a translation rule which depends on the actual expression on which the syntactic rule operates. This means that there are different semantic operations for the various syntactic possibilities. Hence there is a one-many correspondence between the syntactic operations and the semantic operations. Then the mapping from the syntactic algebra to the semantic algebra cannot be a homomorphism. Consequently the framework is not obeyed: the relation indicated in figure 1 by arrow 2 has to be a homomorphism. But also the principle of compositionality itself is violated. In this situation the meaning of the compound expression is not determined by the information which syntactic rule is used and what the meanings of the parts of the expression are, but also information about the actual expressions operated upon is needed. This situation is not a source of great practical problems, since, at least in the examples considered below, the rule can easily be reformulated in such a way that the framework is obeyed.

2.2. Easy to please

This example concerns a variant of Montague grammar proposed in PARTEE 1973. The expressions generated by the grammar contain labelled brackets which indicate the syntactic structure of the expressions. Partee wants to account for the occurrence of verb phrases in conjunctions and infinitives. Examples are given in (1) and (2)

(1) *Few rules are both explicit and easy to read.*

(2) *John wishes to see himself.*

For the production of these sentences a rule called 'derived verb phrase rule' is used. The rule is so close to a correct formulation that I would not like to call it a violation of the framework. It is rather an illustrative slip of the pen.

Derived verb phrase rule (PARTEE 1973)

If $\phi \in P_t$ and ϕ has the form ${}_t[{}_T[he_i]_{IV}[\alpha]]$, then $F_{104}(\phi) \in P_{IV}$, where $F_{104}(\phi)=\alpha'$, and α' comes from α by replacing each occurrence of he_i, him_i, $him_i self$ by he^*, him^*, $him^* self$ respectively.

Examples:

F_{104}(*he$_1$ sees him$_1$self*) = *see him*self*

F_{104}(*he$_7$ is easy to please*) = *be easy to please.*

Translation rule

If $\phi \in P_t$ and ϕ translates into ϕ', then $F_{104}(\phi)$ translates into $\lambda x_i \phi'$.

From the formulation of the translation rule it might not be evident that the translation rule uses syntactic information. But this becomes clear if one realizes that in order to decide what the actual translation is ($\lambda x_1\phi$ or $\lambda x_2\phi$ or ...), one needs to know the index of the first word of ϕ. So syntactic information is used. The correction of this rule is rather simple, in analogy of term-substitution in PTQ we give the syntactic operation an index as parameter: so F_{104} is replaced by $F_{104,i}$. In a later paper (PARTEE 1977a) the rule is corrected in this way.

2.3. The horse Cannonero

DELACRUZ (1976) considers expressions like *the horse Cannonero*. Such expressions belong to a category $\bar{T}$ and they are generated by the following rule:

S3.1 If $\alpha \in B_T$ and $\zeta \in B_{CN}$ then $F_{21}(\zeta,\alpha) \in P_{\bar{T}}$, provided that whenever α is of the form *he$_n$*, $F_{21}(\zeta,\alpha) = \alpha$; otherwise $F_{21}(\zeta,\alpha)$ = *the* ζ α.

Examples:

F_{21}(*horse,Cannonero*) = *the horse Cannonero*

F_{21}(*horse,he$_1$*) = *he$_1$*.

Translation rule:

T3.1 If $\alpha \in B_T$, $\zeta \in B_{CN}$ and α,ζ translate into α',ζ' respectively, then $F_{21}(\zeta,\alpha)$ translates into α' if α is of the form *he$_n$*; otherwise $F_{21}(\zeta,\alpha)$ translates into

$$\lambda P\exists y[\forall x[[\zeta'(x) \wedge \lambda P\lambda z[{}^{\vee}P]({}^{\wedge}\lambda x[{}^{\vee}z={}^{\vee}x])({}^{\wedge}\alpha')(x)] \leftrightarrow x=y]\wedge[{}^{\vee}P](y)]. \tag{3}$$

Translation rule T3.1 depends on the form of the input expressions of the syntactic rule, so it violates the framework. An attempt to formulate the translation rule as a single polynomial in which no syntactic information is used, would require an extension of IL with an if-then-else construction, and with a predicate which discriminates on semantic grounds between variables and constants. I doubt whether the latter is possible. But a simple solution can be found in the syntax. The construction described by Delacruz provides evidence that we should distinguish among the terms the (sub)categories Proper Names and Indexed Pronouns. Rule S3.1 applies only to the category of Proper Names, or alternatively, rule S3.1 is a partial rule which only applies to the subcategory of Proper Names. This approach describes more clearly what the situation is, than the original rule does, or a semantic reformulation would do. A final remark about the formula (3) given by Delacruz. It is not the simplest polynomial expressing the intended semantic operation. I would use instead:

$$(4) \qquad \lambda P \exists y \forall x[\zeta'(x) \wedge \alpha'(^{\wedge}\lambda z[^{\vee}x = {}^{\vee}z]) \leftrightarrow x = y] \wedge [^{\vee}P](y)].$$

3. NON-POLYNOMIALLY DEFINED OPERATORS

3.1. Introduction

The algebra of formulas into which we translate, is obtained from the algebra of IL-expressions by means of restructuring this algebra. This means that new operations may be added, another type structure may be put on the elements, and old operations may be omitted. Following MONTAGUE 1970b, we require that in this process of restructuring, all operations are polynomial operations over IL. This restriction ensures that the interpretation homomorphism for IL determines a unique interpretation for the derived algebra. If one uses operations on IL expressions which are not defined as a polynomial, then there is a great risk of disturbing the homomorphic interpretation. This would mean that we have no interpretation for the derived algebra, thus we are not doing semantics at all! Therefore it is advisable to use only polynomially defined operators.

When we consider examples of operators which are not polynomially defined, it will turn out that in all cases the operator can be replaced by a polynomially defined operator which has the desired properties. Replacement of a non-polynomially defined operator by a polynomially defined one, is (in all these cases at least) a simplification. Thus the requirement of working in accordance with the framework guides us toward a simpler treatment than originally was proposed. This consequence illustrates the heuristic value of the principle of compositionality. So there is, from a practical point of view, no reason to use nonpolynomially defined operators. Theoretical aspects of non-polynomially defined operators will be discussed in section 4.

3.2. John who runs

The approach to natural language followed in BARTSCH 1979 is closely related to the approach followed in the field of Montague grammar. The differences which appear in this and the next example are that the treatment of intensions is different, and that the generated language is somewhat more abstract since it contains brackets and other auxiliary symbols. These differences do not influence the aspect I wish to discuss. Bartsch presents a rule for the formation of term phrases containing non-restrictive relative clauses. Such expressions are formed from a term and a relative sentence by the following rule (BARTSCH 1979, p.45).

S4. If α is a term and β a relative sentence, then $\beta(\alpha)$ is a term. [...].

The corresponding translation rule reads

T4. If α' is the translation of the term α and $RELT(\lambda x\ \beta'(x))$ is the translation of the relative clause β from S4, then $(RELT(\lambda x\ \beta'(x)))(\alpha')$ is the translation of $\beta(\alpha)$, and for all terms α with $\alpha' = \lambda P(\ldots P(\nu)\ldots)$ we have: $(RELT(\lambda x\ \beta'(x)))(\lambda P(\ldots P(\nu)\ldots)) = \lambda P(\ldots\beta'(\nu)\ \&\ P(\nu)\ldots)$.

The translation rule evidently is no polynomial over IL. The rule works well for the translation one usually obtains for term phrases. For *every man* the standard translation is (5), and for this case the rule is perfect.

(5) $\lambda P\ \forall\nu[\mathrm{man}'(\nu) \rightarrow P(\nu)]$.

In case an alphabetical variant of formula (5) is used, the situation changes. Consider (6).

(6) $\lambda Q\ \forall\mu[man'(\mu) \rightarrow Q(\mu)]$.

Translation rule T4 as formulated above does not apply: it is not defined for this representation. Probably we have to be more liberal and consider T4 to be defined for all expression of the indicated form. But there are also formulas which are equivalent to (5) and which are certainly not of the same form. Let R be a variable of the same type as the translation of terms, and consider (7)

(7) $\lambda Q\ \forall\nu[\lambda R[R(man') \rightarrow R(Q)](\lambda P[P(\nu)])]$.

Rule T4 is not defined for this representation. Moreover, application of the rule to the subexpression $\lambda P[P(v)]$ would yield a semantically incorrect result.

This discussion shows the consequence of T4 that it is no longer allowed to exchange logically equivalent formulas. The rule defines a partial function between IL formulas; it is an instruction for formula manipulation, not for compositional semantics. A reaction to these objections against a rule like T4 might be that one adds to the rule a clause stating that in case a formula is not of the mentioned form, it must be reduced to that format. This instruction obscures a lot of problems since it does not say how such a reduction is to be performed. A discussion of the problems arising with this attempt to correct in this way a non-polynomial rule, will be given in section 4.

Can the basic idea of the operation be described in a polynomial way? The desired effect is the replacement of $P(v)$ by $\beta'(v)\wedge P(v)$. This can be obtained giving $\lambda z[\beta'(z)\wedge P(z)]$ as argument of $\lambda P[\ldots P(v)\ldots]$. We must take care furthermore of the binding of the variable P. Thus we come to a version of T4 which is in accordance with our formalisation of the semantic compositionality principle:

T4'. Let α' be the translation of the term α and γ' the translation of the relative clause γ. Then the translation of the compound expression $\gamma(\alpha)$ is:

$$(8) \qquad \lambda Q(\alpha'(\lambda z[\gamma'(z) \wedge Q(z)])).$$

One observes that it is not needed to follow the method hinted at above: to define the intended semantic operator by defining an operator on specially selected representations. The formulation of T4' uses the polynomial expression (8). It is more exact and simpler than the original formulation, and it works well for all formulas equivalent with α' or γ'.

RODMAN(1976) also considers the formation of terms containing a non-restrictive relative clause. He presents a rule which produces such terms out of a term and a sentence, where the sentence contains a suitable variable. His translation rule reads:

If $\alpha \in P_T$, $\phi \in P_t$ and α,ϕ translate into α',ϕ' respectively, then $F_{3,n}(\alpha,\phi)$ translates into $\lambda P\lambda Q[{}^{\vee}P({}^{\wedge}\lambda x_n[\phi'\wedge {}^{\vee}Q(x_n)])]({}^{\wedge}\alpha')$.

This is not the simplest formulation of the polynomial. By one time λ-reduction one obtains

(9) $\lambda Q[\alpha'({}^{\wedge}\lambda x_n[\phi' \wedge {}^{\vee}Q(x_n)])]$.

One observes that this rule is almost identical with the version given above of Bartsch's rule. The differences are due to a different treatment of intensions, and the fact that Bartsch uses the intermediate stage of a relative sentence. Concerning the meaning of the relative clause construction the two solutions are essentially the same. This shows another advantage of the restriction to use only polynomials. It gives us a uniform representation of meanings, and different polynomials can be compared with each other by using known methods.

3.3. Das Mädchen gibt den Apfel dem Vater

BARTSCH (1979) represents a rule which combines an n-place verb-phrase with a term to produce an (n-1)-place verb-phrase. The rule does not in advance fix the order in which the term positions should be filled in: a rule has as parameter a number indicating which position is to be filled. By varying the sequence of 'filling in' one can generate the German versions of *The girl gives the father the apple*, *The girl the apple the father gives*, etc. (the German versions all seem to be parts of correct sentences). The result of substituting the term α on the i-th place of a verb β is indicated by $(\alpha,i)\beta$. The syntactic rule reads (BARTSCH 1979, p.27)

(S1) If β' is a V^n(n-place verb) with the set of n term-places, $K, i \in K$, and if α' is a T(term), then $(\alpha',i)(\beta')$ is a V^{n-1} with the set of term-places $K - \{i\}$.
For this we write $(\alpha'_{T(i)}(\beta' V^n)V^{n-1})$.

(T1) If α'' is the translation of α' as a T, and $\lambda x_j,\ldots,x_m\beta''(x_j,\ldots,x_m)$ with n places, the translation of β' as a V^n, then the translation of $(\alpha',i)(\beta')$ is

$$\lambda x_j,\ldots,x'_i{}'x_i,\ldots,x_m(\alpha''(\lambda x_i(\beta''(x_j,\ldots,x_m)))),$$

with x'_i as the variable that precedes x_i and $'x_i$ as the variable that x_i.

This rule is defined only for special representations of the meaning of the term, and, for reasons related to the ones mentioned in the previous example, it is not acceptable as a rule for compositional semantics. Again the idea behind the formulation of the rule can be formulated by means of a polynomial, thus becoming an acceptable rule and obtaining a shorter formulation:

If α'' is the translation of α' as a T and γ'' is the translation of γ' as a V^n, then the translation of $(\alpha',i)\beta'$ is

$$\lambda y_1,\ldots,y'_i\,{}'y_i,\ldots,y_m(\alpha''(\lambda y_i\gamma''(y_1,\ldots,y_m)))$$

with y'_i as the variable that precedes y_i and ${}'y_i$ as the variable that follows y_i.

3.4. Woman such that she loves him

Below we have the rule for the formation of restrictive relative clauses from PTQ (MONTAGUE (1973)). This rule reads as follows (notation adapted)

S3,n: CN × S → CN

F3,n: Replace he_n in β by *he/she/it* and him_n by *him/her/it*, according to the gender of the first CN in α;
Concatenate (α, *such that*, β)

T3,n: $\lambda x_n[\alpha'(x_n) \wedge \beta']$.

This rule gives rise to an incorrect translation in case β' contains an occurrence of x_n which should not be bound by the λ-operator which is introduced by the translation rule. An example is the production presented in figure 2.

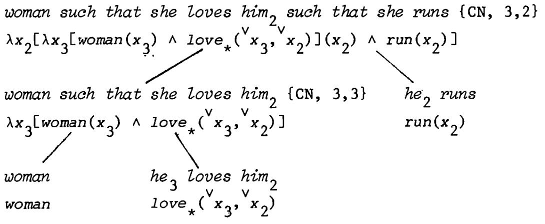

woman such that she loves him₂ such that she runs {CN, 3,2}
λx₂[λx₃[woman(x₃) ∧ love*(ˇx₃, ˇx₂)](x₂) ∧ run(x₂)]
woman such that she loves him₂ {CN, 3,3}
λx₃[woman(x₃) ∧ love*(ˇx₃, ˇx₂)]
he₂ runs
run(x₂)
woman
woman
he₃ loves him₂
love*(ˇx₃, ˇx₂)

Figure 2. Incorrect binding of x_2

The translation obtained reduces to (10).

(10) $\lambda x_2[woman(x_2) \wedge love_*({}^{\vee}x_2, {}^{\vee}x_2) \wedge run(x_2)]$.

The produced CN-phrase may be used in the production of some sentence, and in this process *John* might be substituted for him_2. Then the sentence contains a CN-phrase *woman who loves John*. But the translation contains (10), expressing that the woman loves herself.

In order to avoid this collision of variables, Thomason has presented the following translation rule (footnote to PTQ, THOMASON 1974, p.261, presentation adapted)

T3',n: $\lambda x_m[\alpha'(x_m) \wedge \psi]$

where ψ is the result of replacing all occurrences of x_n in β' by occurrences of x_m, where m is the least even number such that x_m has no occurrence in either α' or β'.

One observes that T3' uses an operation on expressions which is not an operation of IL: the replacement of certain variables by a variable with a special index. We might try to describe the required change by means of IL operators. It is easy to obtain the replacement: λ-conversion does the job:

T3" $\lambda x_m[\alpha'(x_m) \wedge \lambda x_n[\beta'](x_m)]$.

Where m is as in T3'

It is, however, impossible to extend IL with a operator Gr which yields the greatest non-used even index. This can be shown by providing two equivalent formulas for which this operator would yield non-equivalent results. Let ϕ be an arbitrary formula. $Gr(\phi \wedge x_2=x_2)$ would be x_4, whereas $Gr(\phi \wedge x_4=x_4)$ would be x_6, what contradicts the law concerning substitution of equivalents.

We just observed that Thomason's rule contains instructions which essentially use the particular IL representation of the meaning of the relative clause. Nevertheless the rule as a whole is correct in the sense that it corresponds with an operation on the set of meanings. If the translation of the common noun or of the sentence is replaced by an equivalent formula, the application of T3' (or T3") gives for all cases an equivalent result. This is due to the fact that the syntactic operation we called Gr is used only in the context of renaming bound variables. So T3', although violating the framework, does not violate the compositionality principle.

But there is a more direct solution to the problem raised by Montague's rule. A translation rule for relative clause formation which does obey the restriction of using polynomially defined operators is

T3"' $\lambda P \lambda x_n[P(x_n) \wedge \beta'](\alpha')$.

This translation rule yields a correct result for the problematic case presented in figure 2, due to the conditions for λ-conversion. In case α' does not contain free occurrences of x_n, then T3"' reduces to T3, otherwise λ-conversion evokes change of bound variables. One observes that the formulation of T3''' is simpler and much more elegant than the formulation of T3' (or T3"). Moreover T3"' is in accordance with the *variable principle*, whereas T3" and T3' are not (see chapter 8). The simple formulation of T3' is possible because the syntactic problem of collission of variables is not dealt with in the translation rule, but on a more appropriate level: in the rules for λ-conversion which, by their nature, are syntactic operations on IL expressions.

4. OPERATORS DEFINED ON REPRESENTANTS

In all examples from section 3, a rule was defined which works well in the situations one usually meets in practice. In two of the examples the rule does not work well in unusual situations. Often one is tempted to design rules in such a way that as a consequence they have this character. One defines a rule for the formulas one is familiar with, using well known properties of these formulas. Then one hopes that an operation defined on these special formulas in fact defines an operation on the associated meanings. In the present section it will be investigated under which circumstances this hope will become reality. It will be shown that it is not easy to create such circumstances.

Besides the practical motivation given above for considering non-polynomially defined operators, there is a theoretical argument. In the introduction of section 3 I mentioned that an operator which is not defined by means of a polynomial over IL violates the framework, and bears the risk of violating the compositionality principle itself as well. But not all non-polynomial operators do so. In 3.4 we have met a non-polynomially defined operator which could be replaced by a polynomially defined one. From chapter 2, section 7, we know that an operator which is defined on the language IL, and which respects all homomorphic interpretations, can be described by means of a polynomial. But this does not imply that all non-polynomially defined operators which respect the interpretation of IL, indeed can be described by means of a polynomial. This is due to the rather strong condition of the theorem that all homomorphic interpretations are respected. We are not interested in all meaning algebras we might associate with IL, but only in some of them. We want to interpret $P(x)$ as the application of a function to an argument, we want the interpretations of $\phi \wedge \psi$ and of $\psi \wedge \phi$ to be the same, and we want several meaning postulates to be satisfied. Therefore the theorem does not give us the guarantee that every operation on IL which respects the usual interpretations of IL indeed can be defined by means of a polynomial. These considerations constitute a theoretical argument for considering non-polynomially defined operators. But the practical argument given above is, in my opinion a more interesting reason for doing so.

The definition of an operation on IL formulas is acceptable if (and only if) it does not disturb the compositionality principle, i.e. if with the operation on formulas we can associate an operation on meanings. This can only be done if for logically equivalent formulas the operation yields equivalent results. So when defining an operation by defining it for special formulas, every special formula ϕ has to be considered as a representant of the class of formulas equivalent to ϕ.

A mathematical description of the situation is as follows. The set of formulas (of a certain type) is divided into equivalence classes. A class consists of formulas which have the same meaning in all models. Remember that we defined the meaning of an IL formula of type τ as a function which assigns to an index and a variable assignment some element in D_τ (so expressions in the same class represent the same function). In each class representants are defined, and we wish to define an operation on the whole class by defining an operation for the representants. If in each class there is only one representant, we are in the situation presented in figure 2b, if there are more, then we are in the situation of figure 2a. We want to know when a definition on a representant defines an operation on the whole class.

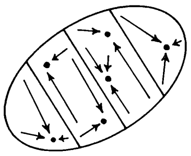
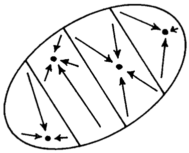



Figure 2a Several representants   Figure 2b One representant

When defining an operation on an equivalence class by a definition on its representant, two aspects can be distinguished.

A) the definition of an operation on the representants

B) A proof that this defines an operation on the whole class.

As for A) we have to fulfill the following two requirements.

A1) One has to describe exactly the set of formulas for which the operation is defined, i.e. one has to define a *recursive* set of representants.

A2) One has to define for all representants what the effect of the operation is, i.e. we have to define an *effective* operation on the set of representants.

This kind of requirements we have met before: define a set and operations on this set (e.g. define a language and a translation of this language). Therefore it seems attractive, in the present context, to define the set of representants by means of a grammar generating the expressions in the subset. In order to be sure that the operation is defined for all formulas in the subset, one might formulate the clauses of the operation parallel to the grammatical rules generating the subset. This will probably be more complicated than a polynomial definition. But other techniques for defining the set of representants and the operation are possible as well.

As for B) I will consider first the situation described in figure 2a: one representant in each class. Here the definition of an operation on the representant automatically determines a semantic operation on the whole class: the interpretation of the operation on the representant is the semantic operation on the interpretations of all expressions in the class. But how can we be sure that we are in the situation of figure 2a? Proving that we are in such a situation means that we have to prove the existence and unicity of a representant for each class. I do not know whether there exists for each type a *recursive* set of unique representants. Assuming the possibility of such a set, it remains the question how to prove the existence and unicity of such representants. It seems attractive to do this by providing an algorithm which transforms a given formula into the representant of its equivalence class. This expresses the idea we met in section 3.2: if a formula is not in the required form, it should be transformed into the required form. This approach is, however, in the relevant cases impossible, as follows from the following theorem.

4.1. <u>THEOREM</u>. *Let* $\sigma,\tau \in$ Ty. *Define the equivalence relation* $\sim$ *on* $ME_{<\sigma,<\tau,t>>}$ *by* $\phi \sim \psi$ *iff* $\models \phi = \psi$. *Let* $R \subset ME_{<\sigma,<\tau,t>>}$ *be a (recursive) set of representants such that for each equivalence class there is one element in* R. *Let* $f\colon ME_{<\sigma,<\tau,t>>} \rightarrow ME_{<\sigma,<\tau,t>>}$ *be a function which assigns to each formula the representant of its equivalence class. Then* f *is not recursive. The* *same result holds if the expressions are from* $ME_{<\tau,t>}$.

<u>PROOF</u>. Let $\phi \in ME_t$, $P \in VAR_\sigma$, $Q \in VAR_\tau$. Then the following statements are equivalent

(11) $\models \phi$ (ϕ is logically true)

(12) $\models \lambda P\lambda Q[\phi] = \lambda P\lambda Q[\forall x[x=x]]$

(13) $\models f(\lambda P\lambda Q[\phi]) = f(\lambda P\lambda Q[\forall x[x=x]])$.

Note that in (13) semantic equality of the formulas is expressed. Since for each class there is one representant, (13) is equivalent with (14)

(14) $f(\lambda P\lambda Q[\phi]) \equiv f(\lambda P\lambda Q[\forall x[x=x]])$.

Note that in (14) syntactic identity of the by f obtained formula is expressed. So, if f is recursive, the question whether ϕ is logically true is decidable: calculate $f(\lambda P\lambda Q[\phi])$ and $f(\lambda P\lambda Q[\forall x[x=x]])$, and see whether they are identical. Since IL is undecidable, f cannot be recursive. For $ME_{<\tau,t>}$ analogously. Note that we did not use the recursiveness of the set of representants.
4.1. END

The translations of expressions of the PTQ fragment are all of the form $<\sigma,<\tau,t>>$ or $<\tau,t>$. The same holds for the expressions of the fragments considered in all examples. The theorem says that there is no algorithm which transforms a formula into its representant. If one tries to define an operation on a class by an operation defined on its representants, then one has to find some other way of proving the existence and uniqueness of a representant.

Next we will consider the situation described in figure 2b: a multiple set of representants is allowed for. There is no doubt that such a set exists: ME_τ itself is a recursive set of multiple representants of ME_τ. But also here a complication arises.

4.2. <u>THEOREM</u>. *Define* $\sim$ *as in the previous theorem. Let* R *be a (recursive) set such that for every equivalence class there is at least one equivalent element in* R. *Let* f: $ME_{<\sigma,<\tau,t>>} \to ME_{<\sigma,<\tau,t>>}$ *be a recursive function that* *assigns to every formula an equivalent formula in R. Then for* $r_1, r_2 \in R$ *it is undecidable whether* $\models r_1 = r_2$.

PROOF. As observed in the proof of the previous theorem $\models \phi$ is equivalent with $\models f(\lambda P \lambda Q[\phi]) = f(\lambda P \lambda Q[\forall x[x=x]])$. If equality is decidable for elements of R, then the equality is decidable for these two formulas, so it is decidable whether ϕ holds. This gives rise to a contradiction since IL is undecidable. Note that we did not use the recursiveness of the set of representants.
4.2. END

This result means that we have to prove that an operation defined for representants yields for equivalent formulas an equivalent result, without knowing what the equivalent formulas look like.

The above discussion shows that it is, generally spoken, a complicated and extensive task to define a function by defining a function on specially selected representations. Probably this can only be done in practice, if one considers a situation with a special structure which has the effect that all proofs become drastically simplified. But if the situation is such a special one, it may be expected that the same effect can be obtained in a more direct way: by using polynomials. This is illustrated in the examples considered in section 3. So far there is no evidence that there is any advantage in defining operations in a non-polynomial way.

5. NEW SYMBOLS IN IL

5.1. Introduction

Some authors extend the language of IL with new symbols. These symbols should obtain an interpretation by means of an extension of the interpretation homomorphism for IL. For each point of reference some semantic object of the right type has to be associated with the new symbol. If the new symbol is an operator, its interpretation has to be a function operating on the interpretation of its argument. If these requirements are not met, then the interpretation homomorphism of IL cannot be extended to an interpretation homomorphism for the extension of IL. Consequently arrow 5 in figure 1 is no longer a homomorphism. Hence arrow 7 is not a homomorphism either. Then the translation homomorphism 2 cannot be continued to an interpretation homomorphism, and this means that the principle of compositionality is violated. Below we will consider two examples of new symbols.

5.2. Shake John awake

DOWTY (1976) treats, among others, the semantics of factive constructions such as *shake John awake*. In order to do so, he extends the language of intensional logic with two operators: CAUSE and BECOME. Interesting for our discussion is his treatment of CAUSE. In order to define its interpretation Dowty adds "to the semantic apparatus of PTQ a selection function f that assigns to each wff ϕ and each $i \in I$ a member $f(\phi,i)$ of I. [Intuitively $f(\phi,i)$ is to be that i' most like i with the (possible) exception that ϕ is the case [..]]". Then the interpretation of CAUSE reads:
"If $\phi,\psi \in ME_t$ then $(\phi \text{ CAUSE } \psi)^{A,i,j,g}$ is 1 if and only if $[\phi\wedge\psi]^{A,i,j,g}$ is 1 and $[\neg\psi]^{A,f(\neg\phi,i),j,g}$ is 1."

The function f is defined on IL-expressions and not on the interpretations of these expressions. As a consequence CAUSE is an operator on IL-expressions and not on the meanings they represent. This is illustrated as follows. The definition of f allows that for some ϕ,η,i holds that $f(\neg[\phi\wedge\eta],i) \neq f(\neg[\eta\wedge\phi],i)$. This may have as a consequence that $[(\phi\wedge\eta)\text{CAUSE } \psi]^{A,i,j,g} = 1$ whereas $[(\eta\wedge\phi)\text{CAUSE } \psi]^{a,i,j,g} = 0$. The main features of an example of such a situation are as follows. Let $[(\phi\wedge\eta)\wedge\psi]^{A,i,j,g} = 1$, so $[(\eta\wedge\phi)\wedge\psi]^{A,i,j,g} = 1$. Suppose that $f(\neg[\phi\wedge\eta],i) = i'$ and $[\neg\psi]^{A,i',j,g} = 1$. Then $[(\phi\wedge\eta)\text{CAUSE } \psi]^{A,i,j,g} = 1$. Suppose moreover that $f(\neg[\eta\wedge\phi],i) = i''$ and $[\neg\psi]^{A,i'',j,g} = 0$. Then $[(\eta\wedge\phi)\text{CAUSE } \psi]^{A,i,j,g} = 0$.

In the above example the principle of compositionality is not obeyed: two equivalent formulas cannot be interchanged 'salva veritate'. Moreover the meaning of CAUSE described above is incorrect since, intuitively, $[(\phi\wedge\eta)\text{CAUSE } \psi]^{A,i,j,g} = [(\eta\wedge\phi)\text{CAUSE } \psi]^{A,i,j,g}$. A correction is possible by taking as domain for f the intensions of formulas: f assigns to each $d \in D_{<s,t>}$ and $i \in I$ a member $f(d,i) \in I$. Then a situation as described above is automatically excluded. The interpreation of CAUSE now becomes as follows.

$$[\phi \text{ CAUSE } \psi]^{A,i,j,g} = 1 \quad \text{if and only if}$$

$$[\phi\wedge\psi]^{A,i,j,g} = 1 \quad \text{and} \quad [\neg\psi]^{A,\underline{i},j,g} = 1, \text{ where}$$

$$\underline{i} = f([{}^\wedge\neg\phi]^{A,i,j,g},i).$$

This definition has the property that if ϕ CAUSE ψ, then for all tautologies η holds that $(\phi \wedge \eta)$CAUSE ψ, a problem of the same nature as the problem we met in chapter 4 concerning the complements of belief-sentences.

5.3. I and You

GROENENDIJK & STOKHOF (1976) give a treatment of the pronouns *I* and *You*. For this purpose, they extend the model for IL. Usually the denotation of an IL expression is defined with respect to a world i and a time j; these i and j are called 'indices'. Groenendijk & Stokhof extend the set of indices with three components: j_0,s and h. Here $j_0 \in J$ is the moment 'now', i.e. the moment of utterance, $s \in A^{I \times J}$ is a function which for a point of reference (i,j) yields the speaker at that moment, and $h \in A^{I \times J}$ is a function yielding the hearer. The interpretation of α may depend on i,j,j_0, s and h, and we write α^{A,i,j,g,s,h,j_0} for the interpretation of α. The language of IL is extended with the constants *i* and *y* of type <s,e>; these constants occur in the translations of the pronouns *I* and *you* respectively. The goal they wish to reach is described as follows. (op.cit.p.308). 'What we want our interpretations to express is that the extension of the constants *i*, *y* are the possible individuals which are speaking now, spoken to now respectively. This would explain the tautological character of a sentence like (15) and the contingent character of sentences like (16)'.:

(15) *I am the speaker*

(16) *I will not be the speaker*

Groenendijk & Stokhof define $F(i) = s$ and $F(y) = h$. Furthermore they define

$$i^{A,i,j,g,s,h,j_0} = F(i)(i,j_0) \qquad (=s(i,j_0))$$

and

$$y^{A,i,j,g,s,h,j_0} = F(y)(i,j_0) \qquad (=h(i,j_0)).$$

So for any point of reference the interpretation of *i* is the speaker now, and the interpretation of *y* is the hearer now.

The corresponding intensions, however, are separately defined: as $(^\wedge i)^{A,i,j,g} = F(i)$ and $(^\wedge y)^{A,i,j,g} = F(y)$ respectively. One observes that no longer holds that for all α: $(^\wedge\alpha)^{A,i,j,g} = \underline{\lambda}(i,j)\ \alpha^{A,i,j,g}$. This combination of the definition of interpretation of $^\wedge i$ and $^\wedge y$ with the interpretation of *i* and *y* violates the recursive interpretation of the IL, thus disturbing its homomorphic interpretation. This has drastic consequences: several tautologies become unvalid. It is no longer true that for constants of type <s,e> holds that $^{\wedge\vee}c = c$, nor that $\alpha = \beta \rightarrow {}^{\vee}\alpha = {}^{\vee}\beta$ is valid (for $\alpha,\beta \in ME_{<s,e>}$). The interpretation of the logic is not a homomorphism (since $h({}^{\wedge}\alpha) \neq \underline{\lambda}i,j[h(\alpha)]$); therefore the interpretation of the natural language is not a homomorphism either. This means that the principle of compositionality is violated.

Let us consider the first goal of the approach of Groenendijk & Stokhof. Sentence (15) is true when evaluated with respect to the moment 'now', but not with respect to a point of reference where the speaker is someone else. The sentence expresses not a tautology (as a matter of fact, this is not claimed by Groenendijk & Stokhof). What they probably wish to express by the phrase 'tautological character' is that for every choice of the moment 'now', the sentence is true when evaluated with respect to this moment, whereas not the same can be said about the second sentence. This effect can be obtained in a compositional way by stipulating that $F(i) = \lambda i,j[s(i,j_0)]$ and $F(y) = \underline{\lambda}i,j[h(i,j_0)]$. Then the translation of (16), being something like ${}^{\vee}i = {}^{\vee}s$, becomes true for the point of reference (i,j_0), no matter what j_0 is, whereas this is not the case for the translation of (16), being something like $\neg W[{}^{\vee}i = {}^{\vee}s]$.

6. COUNTING ELEMENTS

6.1. Introduction

In the present section I will consider two examples of counting the number of elements in the model. In the first example this is done in a way which suggests a misunderstanding of the framework. As a contrast I present the second example in which the counting proceeds correctly. These examples illustrate the role of the derived algebra M' which is obtained from the algebra M in which we interpret intensional logic.

6.2. Keenan & Faltz count

KEENAN & FALTZ (1978) present a system for the description of syntax and semantics that is related to Montague's system. An important difference is that they obtain their semantic domain by application of algebraic operations (join, meet, homomorphism) on certain basic elements. One way in which they compare their system with Montague's is by ways of counting the number of elements in the semantic domain of their system and Montague's sets D_τ. They base an argument in favour of their system on the fact that a certain domain D_τ contains many more elements than the corresponding set in their own system. There are several objections against this comparison. The stage at which Keenan & Faltz carry out their comparison (viz.p.130) does not do justice to Montague's enterprise. They compare their model for an extensional fragment of English with Montague's domains developed for an intensional fragment. Furthermore they do not take Montague's meaning postulates into account. So the numbers they obtain are not the relevant numbers. I will, however, not correct their calculations, since I am primarily interested in the method of comparison. This method will be discussed below.

Keenan & Faltz have a theory which says e.g. how many verb phrase meanings are possible (for a given domain of individuals): it is the number of homomorphisms between certain sets (which are built from the set of individuals). Keenan and Faltz count in their framework the number of elements in some of such sets, i.e. they count the number of possible denotations of certain types. In Montague's framework they count the number of elements in D_τ for the corresponding types. I have fundamental objections against this comparison since in this way sets are compared that are incomparable. The sets D_τ in Montague's system are sets in the algebra M (see figure 1). Out of algebra M a derived algebra is defined. This derived algebra M' consists precisely of the elements which are used for the interpretations of expressions produced by the grammar for the fragment. In the process of forming the derived algebra M' elements of D_τ may be thrown out; e.g. a set D_τ may consist of all functions of a certain type, whereas in M' only the homomorphisms may be left over. If one wants to count the number of possible denotations for the expressions of a certain category, then one has to count the number of elements of the corresponding type in the union of all derived algebras. One should not count the auxiliary set D_τ instead. The method of counting of Keenan & Faltz neglects the role of arrow 6 in figure 1.

The number of elements in a derived algebra can easily be counted. The derived algebra M' is the image of the syntactic algebra for the fragment. Therefore the number of elements in M' cannot be larger than the number of expressions in the syntax. Since the latter is denumerable, the former is. And for a given category, say the verb phrases, the number of expressions of this category gives an upperbound for the number of elements of the corresponding type in the semantic algebra.

6.3. Partee counts

As a contrast to the previous example, I would like to consider PARTEE (1977b), where the difference between M and M' is taken into account. Partee discusses the psychological plausibility of possible world semantics. She argues that the finiteness of our brains requires that the theory of linguistic information we have should be finitely representable. The possible world semantics, however, gives rise to sets of rather large cardinalities (For instance if $|A| = \aleph_0$ and $|I| = \aleph_0$, then $|D_{<s,e>}| = 2^{\aleph_0}$ and $|D_{<<s,e>,<s,e>>}| = 2^{(2^{\aleph_0})}$. These cardinalities make it impossible to assume that we have finite representations of all sets D_τ in our brains. Partee gives a way out of this dilemma: assume that we have finite representations of the form of the sets D_τ, but not of all their elements. PARTEE (1977b,p.317-318) says: 'In the acquisition of any particular language, not all of the in-principle possible denotations need to be considered as a hypothesis about the actual denotation of an actual expression of the language. The intensional logic into which the natural language is translated, will contain at most denumerable many expressions of any given type, and the finite perceptual and cognitive apparatus will make at most denumerable many members of $D_{a,A,I,J}$ finitely representable, and it will only be correspondences between these two at most denumerable sets that will have to be empirically determined by the language learner'.

It is striking to compare these psychologically motivated opinions with the mathematical properties of the framework. These predict that in an interpretation of a particular language there are only denumerable many meanings because there are denumerable many expressions in the language of which we give the meaning. So for any particular language the number of meanings in the model, and the number found on psychological considerations agree.

In relation with the previous discussion Partee considers the following question (PARTEE 1977b p.318). 'One might ask at this point, if the 'available' members of $D_{a,A,I,J}$ are always going to form a denumerable set, why shouldn't all the sets $D_{a,A,I,J}$ be constrained to be denumerable by the semantic theory?' The answer Partee would argue for is 'that there is no telling in advance which possible world the native speaker will find herself in; [..] her semantic component must equip her for a language in any of them.

An alternative is to consider the psychological arguments as an invitation to change one of the algebras of the framework in figure 1. It seems to be an argument against taking the standard model for intensional logic because of the large cardinality of its sets, in favor of taking as semantic algebra some generalized model with denumerable many elements (generalized model in the sense of HENKIN 1950, see sections 1 and 3 of ch.3). This would have the interesting consequence that the axiomatization of intensional logic (given in chapter 3) would be a complete axiomatization for this class of models. This is a direct consequence of lemma 3.3.1 in GALLIN 1975. But no matter which conclusion is drawn from the psychological arguments, this whole discussion remains an interesting excursion because Montague's system was not designed, as I explained in chapter 1 and 3, to formalize any psychological theory.

7. THE TRANSLATION LANGUAGE

7.1. Introduction

An intensional language is a language of which the denotation of an expression depends on an index, and an extensional language is a language where this is not the case. An example of an extensional language is predicate logic, an example of an intensional language is IL. Our approach makes English an intensional language: a sentence denotes a truth value; which one this is depends on the current index (point of reference). In an extensional approach we would say that a sentence denotes an function from indices to truth values (an intension). Is it possible to give an extensional treatment of English, and what are the consequences? In other words, is it possible to change the relation indicated in figure 1 by arrow 7?

It will turn out that the answer to the above question is positive. This gives us the choice between (at least) two different approaches. When making a choice, we have to realize what the role is of the translation level in the whole framework. We aim at defining a systematic relation between English expressions and their meanings. In order to be able to express this relation conveniently, we use the translation into intensional logic. In chapter 2 is explained how this logic is interpreted homomorphically: e.g. an expression of type t has as its homomorphic image (has as its meaning) some function in $\{0,1\}^{I \times J \times G}$. Fundamental to this whole approach is the relation between expressions and meanings. If a translation into an extensional language gives rise to the same relation, it is acceptable as well. Another translation is just another tool, and a choice has to be made on the basis of technical arguments.

7.2. Hausser translates

As an introductory step of treating English as an extensional language, I consider an approach which is very close to the PTQ translation: translate into an IL expression denoting the meaning of that expression. Let the intensionali ed translations of two sentences be A and B respectively. Then the translation of their conjunction has to be $^\wedge[^\vee A \wedge {}^\vee B]$. Most of the translation rules have the format $\alpha(^\wedge\beta)$. With the new translations this becomes $^\wedge[[^\vee A](B)]$. These examples illustrate that this approach does give rise to somewhat more complex formulas. A next step is to use a logical language in which the operators on elements of type $<s,\tau>$ are defined. For instance $\underline{\wedge}$, where $A \underline{\wedge} B$ is interpreted as the complex conjunction formula given above, so as indexwise evaluation of the parts of the conjunction. For function application is used $A\underline{(}B\underline{)}$; to be interpreted as denoting the same as $^\wedge[[^\vee A](B)]$. Such operators are used in JANSSEN & VAN EMDE BOAS 1977 for dealing with semantics of programming languages.

Following TICHY 1971, HAUSSER (1980) argues for an extensional approach to English. In HAUSSER (1979a,1984) this idea is worked out. He does not use the standard logical operators (e.g. conjunction on truth values) any more, and this gives him the opportunity to use the standard symbols with a new meaning. Now $\phi \wedge \psi$ means indexwise valuation of the parts of the conjunction, and $\alpha(\beta)$ is the variant of function application which is described above as $\alpha\underline{(}\beta\underline{)}$. In this way one obtains a simplification of the formulation of the translation rules since no intension symbols have to be used. The price one has to pay for this, is that the logical symbols obtain a somewhat deviant interpretation, what is the normal price, and what is quite acceptable. But the presentation of the translation rules is not the only aspect of a new translation. What happens if one wishes to take the meaning postulates into account, or if one wishes to simplify the formulas one obtains? To understand the dangers, one should realize that the new translation causes the intension operators to be invisible whereas semantically they still are there. Therefore new reduction rules have to be found. In HAUSSER (1979b,1984) indeed a simple translation is obtained but not all meaning postulates are expressed in the translation. Therefore his results are not convincing, and further investigations are required before it is clear whether this extensional approach gives a simplification.

7.3. Lewis translates

A simplification I expect from the approach in LEWIS 1974. He discusses the consequences of another kind of extensionalized translation. He considers using the extra possiblities given by an extensionalized translation: namely the possibility to relate to an expression a meaning that is not an intension (i.e. not a function in $D^{I\times J\times G}$). The verb *run* gets in the PTQ approach a translation of type <<s,e>,t>. In the Hausser approach it is translated into an expression of type <s,<<s,e>t>>. But in the Lewis approach its translation would be of type <<s,e>,<s,t>>. So the translation of *run* would be a function which assigns to an individual concept a proposition. In this way one gets rid of the remarkable non-constant interpretation of constants of IL, where $run^{A,i,j,g} = F(run)(i,j)$. In Lewis approach it would be just $run^{A,i,j,g} = F(run)$. The translation of *He₁ runs* would be $run(x_1)$ being an expression of type <s,t>. Note that here the function application has its standard meaning. This illustrates the advantage of Lewis approach, but further investigations are required in order to decide whether it is a real simplification. A remarkable aspect of Lewis approach is that it gives a completely different relation between expressions and meanings than we considered up till now, so investigating these matters here, might bring us far from the current work (cf. HAUSSER 1984,p.82,83).

7.4. Groenendijk & Stokhof translate

A last version of what might be called an extensionalized translation is used in GROENENDIJK & STOKHOF 1981. They do not translate into IL, but into Ty2 (see GALLIN 1975). Such a translation can be called extensional since the interpretation of a Ty2-expression does not depend on an index, but only on the variable assignment (including the assignment to 'index' variables). Also in this translation we get rid of the non-constant interpretation of constants since the index dependency of predicates as *run* is made explicit by translating *run* into an expression containing an index-variable. The phenomena described by Groenendijk & Stokhof seem to require the expressive power of Ty2, and it is to be expected that this power will be required for the treatment of other phenomena as well (e.g. VAN BENTHEM (1977) argues that explicit reference to moments of time is needed for tense phenomena). Since we will not consider these phenomena, we will not investigate the details of such a translation. From the way in which we introduced IL in chapter 3 (using a translation into Ty2), it is evident that this would cause no problems at all (on the contrary, several aspects would become simplified).

7.5. Keenan & Faltz on translations

In KEENAN & FALTZ 1978, several requirements are given concerning the logical form of a natural language, e.g. criteria concerning the correspondence between a natural language expression and its logical form. They criticize the logical form which is obtained in a Montague Grammar. An example is their comment on the translation of *John* which is in an extensional fragment $\lambda P[P(j)]$. They say (p.18) '... this assignment of logical structure fails the Constituent Correspondence Criterion, since it contains three logical elements, namely j, P and λP, none of which corresponds to a constituent of John.' Such criticism plays an important role in the argumentation in favour of their framework.

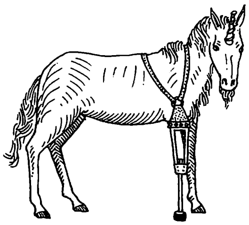
The argumentation of Keenan & Faltz is, however, based upon a misconception of the framework. I assume that they understand by 'logical form', that level of description at which the meaning of an expression is completely determined. In fact, there is no unique level of description in Montague Grammar for which this holds. The analysis tree of an expression, its immediate, unreduced translation, its reduced translation (and all the stages in between), all determine the meaning of that expression completely. That, in particular, the translation of an expression into IL cannot be claimed to have a special status as *the* logical structure of that expression, becomes clear if one realizes that this level of representation, in principle, can be dispensed with altogether. Grammar provides a correlation between syntactic structures and meanings. In Montague Grammar this is done by providing a homomorphism from the set of syntactic structures into the set of abstract settheoretical entities, modelling the meanings. In the PTQ-system this homomorphism is defined in two steps. First a homomorphic translation from syntactic structures into logical expressions is provided, second the logical expressions are interpreted, i.e. related in the usual homomorphic way to the abstract entities defined in the model. These two homomorphisms together determine one homomorphism from the syntactic structures into the meanings, viz. the composition of the two. This two-step approach is chosen for reasons of convenience only, it is not necessary. As a matter of fact, the EFL-system (MONTAGUE 1970a) is an example of a system in which the homomorphism from syntactic structures into abstract meanings is defined in one fell swoop, without an intermediate stage of translation into a logical language. All this means that within the PTQ-framework it is not possible to talk of *the* logical structure, or *the* logical from, of an expression. So Keenan & Faltz criticize a non-existing aspect of Montague grammar.

CHAPTER VII

PARTIAL RULES

ABSTRACT

In the framework the syntactic and semantic rules are considered as algebraic operators. As a consequence of the definitions given in the first chapters, the syntactic rules have to be total. This is investigated and compared with linguistic requirements. Partial syntactic rules from the literature are considered and alternatives for them are presented. One of the methods to avoid partial rules is the use of rule schemes. It turns out that the requirement of using total rules is a valuable heuristic tool. Consequences of this requirement are compared to consequences of Partee's well-formedness constraint.



1. RESTRICTIONS OF THE FRAMEWORK

Based upon the principle of compositionality, we have developed an algebraic framework for the description of syntax and semantics. The algebras of the framework have operators: i.e. functions from carriers to carriers. This implicates that an operator can be applied without any further restriction to any element of the sorts required by the operator. In this chapter I will consider consequences of this aspect of the framework, and especially its consequences for the syntactic algebra. Some of these consequences are closely related with those of the 'well-formedness constraint', (PARTEE 1979b), which will be considered in section 6.

In linguistics one often conceives of a grammar as a generating device for producing all and only the expressions of a language. With this conception it is rather natural to think of restrictions on this production process. One might think of restrictions on the order of application of the rules. Two examples are the following. One might have rules of which the applicability depends on the way in which an expression is produced (such conditions are called 'global constraints'). One might have a filter which throws away some of the produced elements (e.g. one which filters out all expressions which contain a certain symbol). The description of the possible sequences of application of the rules constitutes an important component of a transformational grammar (for instance certain rules might be obligatory, others ordered cyclically), and filters are also often used in that field. If one wishes to use the syntactic knowledge from the field of transformational grammar in the field of Montague grammar, then one is tempted to incorporate these restrictions on the generation process in Montague grammars. Would that be possible, and at what price?

In our framework the syntax has to be a many-sorted algebra, i.e. a set of carriers with operations defined on these carriers. An algebra is not a generating device, it rather is the description of a situation. By describing what the syntactic algebra is, it is said what the relevant expressions are, and what the functions are which describe the relations between these expressions. The expressions can be determined in any way one likes, and nothing has to be said about their production. One might for instance define an algebra by mentioning all the elements and describing the effects of all operators (we did so in the beginning of chapter 2). A simpler method is to give a collection of generators, and tell what the operators are. Several choices of generators may be possible, one more clever than the other. But no matter how the algebra is defined, the elements remain the same elements, the operators remain the same operators, and the algebra remains the same algebra.

The operators of the algebra are mappings from carriers to carriers. The range of an operator (the expressions obtained as results of an application of the rule) consist by definition of elements of a certain carrier. Therefore it is in our framework not possible to have a filter which says that certain outcomes are not acceptable. The domain of an operator (the expressions it operates upon) is some n-tuple of carriers. How we obtained the information that an expression belongs to a carrier is of no influence. The applicability of an operator cannot depend on the information which rules were applied previously, because there are no 'previously applied rules' for an element of an algebra. For this reason, there cannot be a prescribed ordering on the rules, there cannot be rules that are explicitly required to be used in all derivations, and the derivational history cannot influence the applicability of the rules.

Of course, the generation of expressions is an important aspect of syntax, and therefore we paid special attention to it. The notion of a generated algebra was defined, and theorems were proved about such algebras. In a generated algebra it might be meaningful to speak about filtering, ordering of the application of rules, the influence of derivational history, and obligatory rules. But if we would allow this, we would describe a generation mechanism, and not operators of an algebra: in an algebra there is no place for such aspects. So this discussion brings us to reject certain methods which are customary in the tradition of transformational grammars. But the rejection only concerns the method, not the ideas. It is possible to organize an algebra in such a way that the same effects are obtained in another way. Below I will give some examples.

An explicit ordering of rules is not possible in an algebra. But in a generated algebra there is a certain natural ordering among the operators. If an operator R takes as its argument an expression of category C, then the operators which yield expressions of the category C are used before R. In this way the categorial system of the algebra has as effect a certain implicit ordering of the operators.

If one wants a certain ordering on the rules, this effect can be obtained by a suitable refinement of the categorial system. Let R_a and R_b be two rules, both operating on sentences and yielding sentences. Suppose that we want R_a to be applied precisely one time before R_b. This effect can be obtained by distinguishing among the sentences two categories: S_1 and S_2. Here S_1 is the category of sentences to which R_a has not yet been applied, and S_2 of sentences to which R_a has applied. Then R_a can be defined as a rule which operates on expressions of category S_1 and yields expressions of category S_2, whereas R_b is defined to operate on expressions of category S_2, yielding expressions of this category again. The definitions of the other rules have to be adapted for these categories as well. I expect that by means of a refined categorization system the effect of any ordering can be obtained. Since in the field of Montague grammars explicit rule ordering hardly is employed, I will not consider this topic any further.

As explained above, the applicability of a syntactic rule to an expression cannot depend on the derivational history of that expression. Notice that, on another level, we already met a situation where it was important to have derivational histories available. The meaning of an expression may depend on the derivational history of that expression. We did not define the translation homomorphism on the algebraic grammar for a language because in that grammar such histories are not available. The translation homomorphism is defined on the associated term-algebra, i.e. the algebra of derivational histories. This suggests us what to do when derivational histories would be important in syntax: use an algebra in which the elements represent derivational histories. But in the field of Montague grammars I know of only one rule which uses information about the derivational history (rule 3 of THOMASON 1976), so the issue does not seem to be important. Moreover, this aspect of Thomason's rule can probably be avoided by following the proposal of PARTEE (1973) to let a grammar produce not unstructured strings, but labelled bracketings. For these reasons the role of derivational histories in the syntax will not be considered here any further.

Above we have considered some restrictions on the circumstances in which a rule may be used. The conclusion was that such rules violate basic aspects of our framework. Another request from linguistics is to allow restrictions on the expressions to which a rule is applied. In the field of transformational grammars it is standard to put conditions on the possible inputs of a transformation. In the field of Montague grammar many rules are proposed as well which do not apply to all expressions of the category required be the rule, but only to some of them. In the field of semantics one has proposed to use functions which are not defined for all arguments of the required type (see section 2.2). In contrast to the constraints on applicability discussed above, one might argue that our framework should allow for operators which are not defined for all arguments of the required sort. Such partial operators are known in the theory of universal algebras; the algebras in which they occur are called partial algebras. In the next sections it will be investigated whether we could be more liberal than we have been, and whether we should allow for partial algebras within our framework.

2. PARTIAL ALGEBRAS

2.1. Partial grammars

Contrary to what one might expect, it is not just a minor variation of the system to allow for partial algebras (i.e. algebras with partial operations). Such a step would disturb important parts of theory we have developed so far. I will illustrate this by means of two examples which show that certain theorems we proved concerning properties of the syntax are not valid when partial rules are allowed. In 2.2 it will be shown that certain theorems of intensional logic loose their validity when partial operators are allowed in the logic.

2.1. EXAMPLE.

$$G = \langle\langle[\{a_A\},\{b_B\},\{c_C\}],\ \{F_a,F_b,F_c,F\}\rangle,D\rangle.$$

Here F_a: A → A is defined by $F_a(\alpha) = \alpha a$

F_b: B → B is defined by $F_b(\beta) = \beta b$

F_c: C → C is defined by $F_c(\gamma) = \gamma c$.

So by repeated application of F_a strings of arbitrary length consisting of a's are produced. Analogously for F_b and F_c. Furthermore the partial rule F is defined as follows:

$$F: A \times B \times C \to D$$

where

$$F(\alpha,\beta,\gamma) = \begin{cases} \alpha\beta\gamma & \text{if the lengths of } \alpha,\beta \text{ and } \gamma \text{ are equal.} \\ \text{undefined} & \text{otherwise} \end{cases}$$

The language L(G) generated by G is $\{a^n b^n c^n \mid n \in \mathbb{N}\}$. This is a non-context-free language (see HOPCROFT & ULLMANN 1979 example 6.1). So when partial operations are allowed in the syntax, theorem 5.6 from chapter 2 does not hold.

2.2. <u>EXAMPLE</u>. Let L be some recursively enumerable language over alphabet A. According to theorem 3.7 from chapter 2, there is an algebraic grammar G such that L(G) = L. Suppose that $G = \langle\langle [B_s]_{s\in S}, (F_\gamma)_{\gamma\in\Gamma}\rangle, s_0\rangle$.
Let $\sigma \in A^*$ be arbitrary, and define the algebraic grammar H_σ by
$H_\sigma = \langle\langle [B_s]_{s\in S}, (F_\gamma)_{\gamma\in\Gamma}\cup\{f\}\rangle, s_1\rangle$
where s_1 is a new sort ($s_1 \notin S\cup\{s_0\}$), and where f is a partial operation defined by

$$f\colon s_0 \to s_1 \text{ where } f(\alpha) = \begin{cases}\sigma & \text{if } \alpha \equiv \sigma\\ \text{undefined} & \text{otherwise.}\end{cases}$$

Note that H_σ produces a language which is either empty (if $\sigma \notin L(G)$) or consists of σ (if $\sigma \in L(G)$). So $L(H_\sigma) \neq \emptyset$ iff $\sigma \in L(G)$.

Suppose now that it was decidable whether $L(H_\sigma) = \emptyset$; then it was decidable as well whether $\sigma \in L(G)$. Since L(G) is an arbitrary recursively enumerable language, the latter is not decidable, and consequently it is not decidable whether $L(H_\sigma) = \emptyset$. This means that theorem 5.5 from chapter 2 (which states the decidability of the emptiness of L(G)) is not valid if we allow for partial operations.

2.2. <u>Partial models</u>

The following example concerns the use of partially defined operations in the semantics. They arise, for instance, if one wishes to deal with sortal incorrectness: certain combinations of a verb phrase with a subject do not fit well together, although most expressions of their categories give no rise to problems. An example (THOMASON 1972) is (1).

(1) *The velocity of light is shiny*
It is not attractive to say of such a sentence that it is 'false', since then its negation would be 'true'. Either, one should consider (1) as being syntactically incorrect, or the strangeness should be dealt with in the semantics. THOMASON (1972) followed the latter approach and has proposed

to assign to such sentences no truth values. This idea is worked out in the framework of Montague grammar by WALDO (1979). In his proposal several semantic domains contain partial functions, and the function corresponding with *shiny* is not defined for arguments such as 'the velocity of light'. So (1) is not associated with a truth value.

Waldo's approach gives rise to strange consequences. Formulas which one might expect to be equivalent, are not. I will discuss two examples, and indicate how the problems could be solved by using total functions in the model.

The first example concerns formula (2), where $\phi \in ME_t$.

(2) $\phi = \phi.$

Suppose that the interpretation of ϕ is undefined (e.g. because it is the translation of (1)). Then, due to the interpretation of =, also (2) is undefined. However, due to the interpretation of connectives (which uses 'extended interpretations'), formula (3) gets the interpretation *true*: This difference in interpretation is, in my opinion, a strange result.

(3) $\phi = \phi \wedge \phi = \phi.$

The second example is based upon a suggestion of R. Scha (pers.comm.). It concerns formula (4), where $z \in VAR_t$, and where $\phi \in ME_t$ is as in (2).

(4) $\lambda z[z=z](\phi).$

The possible assignments to z are, in Waldo's partial model, the truth values *true* and *false*. Therefore the expression $z = z$ is equivalent with some tautology not containing z, for instance $\forall w[w=w]$. Hence (4) is equivalent to (5)

(5) $\lambda z[\forall w[w=w]](\phi).$

According to the standard conditions for λ-conversion, formula (5) can be reduced to (6), which clearly gets the interpretation *true*.

(6) $\forall w[w=w].$

Also in (4) λ-conversion is, according to the standard conditions, an allowed operation. Then (2) is obtained, but the interpretation of that formula is undefined. So the formulas (6) and (2), obtained by reduction of (4) are not equivalent, an unacceptable result. We have to conclude that one of the reductions steps is not allowed. This problem is, in my opinion due to the fact that for the variable z in (4), there are two possibilities (true and false), whereas for the arguments ϕ there are three possibilities (true, false, and undefined). Note that the variable z cannot be undefined, because its range consists of all elements in the model of the correct type, and *undefined* is no value in the model.

The above examples show that the laws of logic we have met before, cannot be used in this system without further investigations. In any case the conditions for λ-conversion have to be changed. Unfortunately, Waldo does not provide laws for his system. This causes a difficulty in the study of his proposal. He presents several examples, and each consists of a sentence accompagnied by its reduced translation. Since I do not know which reduction rules hold in an approach with partial functions in the semantics, I cannot check the correctness of the examples. Also other authors who describe a fragment with the use of partial functions in the semantic domains, do not present reduction rules (HAUSSER (1976), COOPER (1975)). The last author mentions at least that not all standard reductions remain valid. I expect that it will be very difficult to reformulate the reduction rules. An obvious attempt to improve the conditions for λ-conversion would be to require that the reduction of $\lambda z[\alpha](\beta)$ is allowed only if $\beta$is defined. This is, however, not a syntactic condition on β, and I doubt whether it is possible to give a syntactic characterization of 'undefined'.

I already explained that the problem is due to the fact that a variable cannot take the value *undefined*, whereas an argument ϕ (which might be substituted for that variable) can be undefined. Therefore I expect that the problems will be solved when a third truth value is introduced, say a value *error*. In any case, the two problems mentioned above disappear. If the value *error* is assigned to z, then the interpretation of $z = z$ is always the same as the interpretation of $\phi = \phi$, even in case ϕ is undefined. Now λ-conversion is allowed both in (4) and in (5), and furthermore, all formulas (i.e. (2)-(6)) get the same interpretation for all values of ϕ. Note that this plea for using a third value is not an argument for using some of the existing tables for three valued logic. Waldo uses supervaluations (Van FRAASSEN 1969), and one might try to reformulate supervaluations for an approach with a third truth value.

The idea of using a third truth value is not new; it goes back to ŁUCKASIEWICS (1920), who gives tables for the connectives in a three-valued system. In the theory of topoi one introduces a value representing 'undefined' (GOLDBLATT 1979, p.268). In the theory of semantics of programming languages the problems of working with 'undefined' are well-known. Undefinedness arises, for instance, when a process is defined for calculating the value of a function, whereas the process does not terminate normally because not enough memory capacity is available. The standard approach in this field is not to use partial functions, but to make the functions total by means of the introduction of an extra element in the semantic domain, called 'errorvalue' or 'bottom' (SCOTT & STRACHEY 1971, GOGUEN 1978). In the field of Montague grammars the situation is as follows. A model for intensional logic with 'undefined' as possible value, is presented in Von KUTSCHERA (1975). It is however common practice to consider undefinedness not as a value (see KAMP 1975, COOPER 1975, HAUSSER 1976, WALDO 1979, KLEIN 1981). I know of only one author who presents a treatment of a certain fragment and uses a model with 'undefined' as value: Ter MEULEN (1980).

2.3. Discussion

The examples given in sections 2.1 and 2.2 show that it will be a considerable change of the framework to allow for algebras with partial operations. Moreover, it is not obvious in which way we have to proceed. GRAETZER (1968,p.80) says the following. 'For algebras there is only one reasonable way to define the concepts of subalgebra, homomorphism, and congruence relation. For partial algebras we will define three different types of subalgebra, three types of homomorphism, and two types of congruence relation .[..] all these concepts have their merits and drawbacks'. This situation constitutes an argument for my expectation that it will be a considerable task to develop a framework based upon the use of partial algebras. What I have seen of the literature concerning partial algebras did not give me the confidence that an elegant framework can be built using this notion (e.g. ANDREKA & NEMETI (1982), MIKENBERG (1977), ANDREKA, BURMEISTER & NEMETI (1980)). The example concerning partial functions in the semantics gives me the conviction that it is not a good idea to base a semantic theory on partial functions. For these three reasons I do not sympathize with the idea of basing the framework on partial algebras.

As for the introduction of partial rules in the syntax only, the situation seems to be different. It is just a minor change of the framework because the homomorphic relations between the algebras of the framework are hardly disturbed. An argument in favor of the introduction of partial rules is that such rules are used frequently in practice. But there also are arguments against the introduction of partial rules in the syntax. Below I will mention some of them, thereafter they will be discussed.

1. *Consistency of argumentation*

The first argument concerns the consistency of our argumentation. In a Montague grammar we distinguish categories, and the rules give the information in which way the expressions of certain categories combine to form expressions of other categories. An argument for distinguishing such categories (given e.g. in chapter 1) was that certain groups of expressions behave differently from other groups in syntactic or semantic respects. Designing partial rules would mean that among a single category we distinguish two subgroups (these expressions of a category to which the rule can be applied, and those to which the rule cannot be applied). A consistent reaction in such a situation would be to conclude that the system of categories was not refined enough, and that the system has to be refined in such a way that the partial rules are no longer partial.

2. *Filtering power*

A partial rule introduces a kind of filter in the grammar, and filters form a powerful tool which can easily be abused. In a Montague grammar the syntactic rules provide the information which kinds of expressions may be combined to form new expressions. But partial rules would make it possible that the syntactic rules combine rubbish to rubbish, whereas a final partial rule would filter out the undesired expressions. In this way, the other rules would not give information about the combinations which make sense and which not. The filtering power of partial rules in syntax is employed in the first two examples given above.

3. *Generation of expressions*

Often one wishes to conceive a grammar as a generating device. The rules of the fragment presented in chapter 4 can easily be conceived in this way. A rule like S_4 is considered as an instruction stating that if one wants to generate a sentence, one has to generate a term and an

IV-phrase, and next combine them. The rules for term-formation and IV-formation are, in the same way, considered as instructions for a generating process. The processes of generating a term and of generating an IV-phrase may be carried out independently, and every outcome of the processes is acceptable. Details of a computer program based upon these ideas can be found in JANSSEN (1980a). Suppose now that the grammar contains a partial variant of S_4, say a rule which imposes restrictions on the possible combinations of a term with an IV-phrase. Then the simple algorithm just sketched cannot be used. One has to design an algorithm that gives the guarantee that after a finite amount of time an acceptable combination is found (provided that there is one). This requirement would make the algorithm rather inefficient: the only possibility I see for such an algorithm is one which tries out all possible combinations of a term with an IV-phrase. So in the perspective of a generation process partial rules are unattractive.

4. *Consequences*

An important argument in favor of total rules is that this requirement has attractive consequences. On a more theoretical level it gives rise to an interesting restriction on the possibility to incorporate transformations in a Montague grammar (see section 3). On a more practical level the requirement of using total rules turns out to be a valuable heuristic tool. Several partial rules from the literature can be reformulated or eliminated, and the requirement suggests how this can be done. Thus several proposals from the literature can be replaced by a simpler treatment (see section 4).

5. *No theory*

The introduction of partial rules, even if only in the syntax, constitutes a considerable change of the framework. As the given examples have shown, the theory which we have developed, cannot be used without corrections. Since a theory about partial syntactic algebras is not available, there is no guarantee that all consequences are acceptable.

None of these five arguments is decisive. As for 'consistency', it is indeed more elegant to use the argument for the introduction of categories in all situations with the same conclusion. But with respect to other considerations there might be arguments of elegance in favor of partial rules (e.g. that in that way linguistic generalizations can be captured). That partial rules introduce a powerful filter, is not an impressive theoretical argument since the algebraic grammars have a universal generative capacity anyhow. As for the argument of 'generation', it is not a primary aim of our grammar to develop an efficient generating device. From a practical point of view, a parser might even be of more importancy than a generator. The fact that the practical consequences of using total rules turns out to be attractive in the situations considered, is not a guarantee that in other cases this will be the case as well, and that there is no theory about partial algebraic grammars might be a challenge to develop such a theory.

The arguments against the introduction of partial rules and the arguments in favor of doing so, have to be weighed against each other. The arguments given above show that there are several unattractive aspects related with the introduction of partial rules. I do not know of convincing arguments for the need or attractiveness of partial rules. In the remainder of this chapter I will show that there are several alternatives for the introduction of partial rules. These alternatives are: reformulating as a total rule (section 3), reformulating as a rule operating on another category (section 4) and a refined system of subcategories (section 5). It will turn out that the use of these alternatives gives, in most cases, rise to a simpler treatment than originally proposed: the requirement of using total rules turns out to be a valuable heuristic tool. So the situation can be summarized as follows: there are arguments against the introduction of partial rules, and attractive alternatives are available. Therefore I do not feel enthousiastic about the introduction of partial rules in the syntax. I do not state that I will never use partial rules myself, but I would first try to use total rules.

3. INCORPORATING TRANSFORMATIONS

In the field of transformational grammars, the use of partial rules is standard. As part of their specification the transformations always contain a restriction on the inputs to which they may be applied (a SC: i.e. structural condition). One might wish to incorporate transformations in Montague grammar in order to benefit from the syntactic insights obtained in that field. In this section I will present a general method for the incorporation of a class of transformations in a Montague grammar in which all rules have to be total.

Some characteristics of transformations are as follows

1. Transformations define mappings from trees to trees; these trees represent constituent analyses of sentences.
2. If several transformations can be applied, then their order of application may be prescribed.
3. A transformation is applied to one input tree at a time.
4. A transformation imposes structural conditions determining the possible input trees.

In order to take care of the first point, it is required that a Montague grammar does not produce plain strings, but trees, (or, equivalently, labelled bracketings). Let us assume that Montague's framework is adapted in the way proposed by PARTEE (1973). So the grammar produces trees. This change of the system turns all rules into rules which operate on trees, so in a certain sense all the rules in the grammar become transformations. In order to avoid confusion of terminology, I will use the name C-transformation ('Chomskyan') for transformations used in transformational grammars. Once that they are incorporated in a Montague grammar, they are called M-transformations.

The second characteristic point is not acceptable in our framework. As explained in section 1, explicit rule ordering does not fit into the algebraic approach. But an implicit rule ordering which has the same effects might be possible. The third point does not give rise to problems. Although the rules in a Montague grammar mostly take two arguments, there is no objection against rules taking one argument. The fourth point is problematic since it implies that C-transformations are partial rules. This is an important characteristic of C-transformations which makes them very attractive for practical use. It makes it possible to indicate in a simple way what the relevant input trees are, without the need to bother about irrelevant inputs.

I will incorporate a class of C-transformations in a Montague grammar which requires total rules, by reformulating them in a way which makes them total. The reader might be surprised by this reformulation and at first glance consider it as a sneaky trick employed in order to obey the letter of the principle. This is not completely true. The reformulation expresses a different view on transformations than the standard one, and it has interesting consequences.

The reformulation proceeds as follows. Suppose that a C-transformation is given in the following form.

> *If the input sentence satisfies structural condition* SC, *then we may apply transformation* T *in order to obtain a new sentence, otherwise* T *cannot be applied.*

Its reformulation as a total rule has the following form.

> *To the input sentence we apply operation* T'. *Operation* T' *is defined as follows. If the input sentence satisfies the structural condition* SC, *then transformation* T *is applied, and otherwise the 'do nothing' transformation is applied.*

By the 'do-nothing' transformation is understood an operation which does not produce another expression, but which gives its input unchanged as output. The reformulation expresses the view that an M-transformation applies to all expressions of the required category, and that its application yields always a result.

Corresponding with a M-transformation T' there has to be a translation rule τ. For the cases that we did 'nothing' in the syntax, we do 'nothing' in the semantics: the input formula is given unchanged as output. This means that for these cases the translation rule τ can be represented as the polynomial $x_{t,1}$. Since in our framework τ has to be represented by means of a single polynomial expression, τ yields for each input formula, that formula as output. So the M-transformations (obtained with the method described here) do not change meanings. Consequently, if one wants to incorporate C-transformations in this way in a Montague grammar, then these transformations have to be meaning preserving! This requirement is a well-known hypothesis in the so called standard theory of transformational grammars (see PARTEE 1971 for a discussion); it is, however, nowadays not generally accepted.

The conclusion that transformations have to be meaning preserving, holds only for the method described above. But I do not know of any other uniform method for incorporating transformations in a Montague grammar with total rules. To illustrate this, I consider one attempt. Instead of requiring that the translation rule corresponding with a do-nothing transformation is the identity operation on formulas, we might require that it is the identity operation on meanings (but not necessarily the identity on formulas). This would make it possible that the polynomial is not the identity when interpreted for the real transformation. Such a rule τ has the following effect:

$$\tau(\phi) = \begin{cases} \rho(\phi) & \text{if } \phi \text{ is the translation of an expression which satisfies the conditions for application of the transformation } (\rho \text{ formalizes the semantic effect of the transformation}) \\ \phi & \text{otherwise.} \end{cases}$$

The first objection is that this effect cannot be obtained by means of polynomial over IL. In order to obtain the effect of such a choice, IL has to be extended with something like the *if-then-else* construction. There would be, however, no problem in doing so. A more essential objection is that in the description of the translation rule τ information about the (syntactic) expressions is used. This has to be replaced by information concerning their meanings. For most transformations there is probably no semantic property corresponding to the condition on the transformation. In any case, we have no uniform method for obtaining a logical condition which is equivalent with the structural condition of the transformation. So a uniform method for finding the polynomial cannot be given.

I described a uniform method for the incorporation of a class of transformations in Montague grammar by means of a do-nothing transformation. This method might be generalized to a method to eliminate certain partial rules from a Montague grammar. For rules with one argument the method can be used if the rule is meaning preserving. For rules with more than one argument the use of a kind of do-nothing transformations implies that (at least) one of the inputs should have the same category as the output. The do-nothing transformation has to correspond to a translation rule which is the identity on formulas. Therefore the translation rule which corresponds with the original partial rule has to be the identity translation for one of its arguments. So this method can be used only for very limited class of the partial rules with more than one argument.

4. DEFINED FOR ANOTHER CATEGORY

4.1. Introduction

In this section I will consider several rules from the literature which are partial, and for which the corresponding translation rule is not meaning preserving. This implicates that the method developed in the previous section cannot be used for them. The method employed in this section is to reformulate the rule for another category than where it was originally formulated for. It turns out that in all cases the new version of the rule is simpler than the original formulation of the rule, and sometimes the original rule was incorrect whereas the new rule is correct. This shows the heuristic value of the framework, and of the requirement of using total rules in particular. The examples are presented in the notation of the original proposal; most examples were already mentioned in JANSSEN (1978a).

4.2. He$_1$ is loved

PARTEE (1973) considers the M-transformation 'Passive Agent Deletion'. An example is

$$F_{102}(\textit{he}_1 \textit{ is loved by } \textit{him}_3) = \textit{he}_1 \textit{ is loved.}$$

Translation:

If $\phi \in P_t$ and ϕ translates into ϕ', then $F_{102}(\phi)$ translates into $\exists x_j \phi'$.

On the one hand this transformation applies only to input trees of a special form, on the other hand the translation rule is not the identity mapping. This means that we cannot reformulate this transformation as a total rule, and that Partee's way of dealing with agentless passive is disallowed by the requirement of using total rules. For the example under discussion, the literature provides an alternative. THOMASON (1976) presents rules for generating passive directly, i.e. without a passive transformation and without a passive agent deletion.

4.3. Give John a book

The C-transformation of dative shift changes sentence (7) into (8).

(7) *Mary serves the cake to John*

(8) *Mary serves John the cake.*

A refined category system for sentences in which dative-shift would be a total rule is very difficult to design (since each new subcategory would require rules producing expressions of that subcategory). Also here the literature contains an alternative. DOWTY (1979a) shows that the partial rule of dative shift on the level of sentences, can be replaced by a rule on the level of lexical elements. That rule changes the category of the verb *serve* from DTV (verbs which take a Dative and a Term) to TTV (verbs which take two Terms). By having a sufficiently refined category system, these lexical rules become total rules. Many examples of transformations which are reformulated on the lexical level can be found in DOWTY 1978, 1979a, and in BARTSCH 1978b, thus they can easily be reformulated as total rules.

4.4. Mary shakes John awake again

In chapter 5, section 5.2, we considered some semantic aspects of the proposals of DOWTY (1976) concerning the treatment of factives. Now I will consider some syntactic aspects (of course, in doing this, the semantic aspects cannot be neglected). Dowty produces the factive sentence *Mary shakes John awake* from the term *Mary* and the IV-phrase *shake John awake*. This IV-phrase in its turn is obtained from the TV-phrase *shake awake*. The first rule Dowty presents for generating this TV-phrase is as follows.

S_{30}: If $\alpha \in P_{IV}$ and $\phi \in P_t$ and ϕ has the form he_n *is* γ
then $F_{30,n}(\alpha,\phi) \in P_{TV}$ where $F_{30,n}(\alpha,\phi) = \alpha\gamma$.

An example is:

$F_{30,1}$(*shake*, *he*$_1$ *is awake*) = *shake awake*.

The corresponding translation rule reads:

T_{30}: If α translates into α' and ϕ
translates into ϕ' then $F_{30,n}(\alpha,\phi')$ translates into:
$\lambda P\lambda x\,{}^{\vee}P({}^{\wedge}\lambda x_n[\alpha'(x,{}^{\wedge}\lambda P[{}^{\vee}P(x_n)])\ \text{CAUSE}[\text{BECOME}[\phi']]])$.

This rule is a partial rule which is not meaning preserving, so we have to find another approach. Can the above result be obtained by means of a total rule? For generating expressions like *shake awake* one only needs an adjective and a TV-phrase. So it lies at hand to try the following rule

S_{601}: If $\alpha \in P_{TV}$ and $\beta \in P_{adj}$ then $F_{601}(\alpha,\beta) \in P_{TV}$ where $F_{601}(\alpha,\beta) = \alpha\beta$.

The corresponding translation rule would be

T_{601} = If α translates into α' and ϕ translates into ϕ' then $F_{601}(\alpha,\beta)$
translates into
$\lambda P\lambda x[{}^{\vee}P({}^{\wedge}\lambda y[\alpha'(x,{}^{\wedge}\lambda P[{}^{\vee}P(y)])\ \text{CAUSE}[\text{BECOME}(\beta'(y))]])]$.

Why did Dowty propose a production of *shake awake*, with as intermediate stage the sentence *He*$_1$ *is awake?* This has probably to do with the ambiguity of *Mary shakes John awake again*. On the one reading Mary has done it before, on the other John has been awake before. Dowty treats *again* as a sentence modifier and he needs two different sentences in the derivation in order to deal with the ambiguity. He starts his investigations along this line probably for historical reasons: it is the way in which such constructions are treated in generative semantics. But, as in the previous examples, we need not to follow the old pattern. By rule R_{601} we are guided to another approach to this ambiguity. The one reading can be obtained by combining *again* with *Mary shakes John awake*, the other by combining it with *shake awake*. I do not go into details of this approach for the following reason. After considering several phenomena concerning factives, Dowty observes that his first approach is not completely adequate. He discusses extensively several alternatives and escapes. Finally he concludes 'there would be no reason why we should not then take the step of simplifying rules S30-S32 drastically by omitting the intermediate stage in which a sentence is produced'. Next he presents as the rule which he considers as the best one, a rule which is identical with S_{601}. So the framework has led us immediately to the solution which is the simplest and best one. This example suggests that we might derive from the framework the advice 'when designing a syntactic rule, ask for what you need as input and not for more'.

4.5. <u>See himself</u>

In chapter 5, section 2.1, we considered the derived verb phrase rule of PARTEE (1973). This rule makes verb phrases out of sentences. An example is

$$F_{104}(he_1 \text{ sees } him_1 \text{ self}) = \text{see } him^*self.$$

The syntactic part of this rule reads as follows:

If $\phi \in P_t$ and ϕ has the form ${}_t[{}_T[he_i]_{IV}[\alpha]]$, then $F_{104}(\phi) \in P_{IV}$, where $F_{104}(\phi) = \alpha'$, and α' comes from α by replacing each occurrence of he_i, him_i, $him_i self$ by he^*, him^* him^*self respectively.

At the one hand the derived verb phrase rule is a partial rule, at the other hand its output belongs to a different category than its input. Therefore we cannot reformulate this rule as a total one using a do-nothing transformation. The derived verb phrase rule is disallowed by the requirement of using total rules, and we have to find another treatment for the cases where Partee uses this rule. Let us, in accordance with the advice given in section 4.4, just ask for what we actually need and not for more. In the above example we only need a TV-phrase. So we might try the following rule.

S_{602} If $\alpha \in P_{TV}$ then $F_{602}(\alpha) \in P_{IV}$ where $F_{602}(\alpha) = \alpha$ *him*self*.

The corresponding translation rule reads:

T_{602} If α translates into α', then $F_{602}(\alpha)$ translates into $\lambda x[\alpha'(x, {^\wedge}\lambda P[{^\vee}P(x)])]$

Would this rule be an acceptable alternative?

Let us consider why one would like to generate *see himself* from the source sentence *he sees himself*. There are semantic arguments for doing so. The sentence *John sees himself* is obviously semantically related to the sentences *John sees John* and *He$_1$ sees him$_1$*. In transformational grammar this might be an argument for producing these sentences from the same source: no other formal tool is available. The effect of Partee's rules is that such a transformation is split up into several stages; it amounts to the same relations. Montague grammar has a semantic component in which semantic relations can be laid formally. So if we do not have to ask for a sentence as source for syntactic reasons, we are not forced to do so on semantical grounds. So this cannot be an argument against S_{602}.

PARTEE (1975) provides as an explicit argument for the derived verb phrase rule the treatment of sentence (9)

(9) *John tries to see himself.*

This sentence is generated, using the derived verb phrase rule, from sentence (10)

(10) *he$_3$ tries to see him$_3$self.*

The translation of (9) becomes in this case (11)

(11) $try\ to({^\wedge}john, {^\wedge}\lambda x_3[see(x_3, {^\wedge}\lambda P[{^\vee}P(x_3)])])$.

Sentence (9) can also be generated according to the rules of PTQ. If we do not change the syntactic details of the rule the following sentence is produced:

(12) *John tries to see him.*

In (12) *him* is coreferential with *John*. The translation is

(13) $try\ to(^{\wedge}John, ^{\wedge}see(\lambda P[^{\vee}P(^{\wedge}John)]))$.

Partee provides arguments for her opinion that interpretation (11) might be preferable to (13). Let us assume that her arguments hold and consider whether S_{602} is compatible with that. The combination of *try to* with the translation of *see him*self* (obtained by T_{602}) yields

(14) $try\ to(^{\wedge}\lambda x[see(x, ^{\wedge}\lambda P[^{\vee}P(x)])$.

So the translation of *John tries to see himself* is, as desired, equivalent to (11). As Partee notices, the derived verb phrase rule does not prohibit the unwanted reading (13). Rule S_{602} is an improvement since it only allows for reading (11). Of course, S_{602} does not give a complete treatment of reflexives, and I am not sure whether I would like to treat them in this way. For the purpose of the discussion this aspect is irrelevant: I just would like to demonstrate that the requirement of using total rules, and in particular the advice 'ask for what you need', guides us to a better rule than originally proposed.

4.6. Easy to see

PARTEE (1975) presents another example for the derived verb phrase rule:

F_{104}(*he*$_7$ *is easy to please*) = *be easy to please*.

This example may seem somewhat strange since it produces the IV-phrase *be easy to please* from a sentence containing this IV-phrase. The reason is that the sentence is obtained by some transformation from the source

(15) *To please him*$_7$ *is easy*.

This transformation is not sufficient for producing all sentences containing the phrase *be easy to please*. Phrases resulting from F_{104} have to be produced as such, in order to generate (16) and (17).

(16) *few rules are both explicit and easy to read*

(17) *try to be easy to please*.

In PARTEE (1977a) other constructions are considered which contain expressions of this kind, such as

(18) *John is being hard to please*.

In order to deal with such expressions Partee needs another rule, called the derived adjective rule, which has the following effect

$$_{IV}[\textit{be easy to please}] \rightarrow {}_{ADJ'}[\textit{easy to please}].$$

This is again a partial rule which cannot be brought in accordance with the restriction of total rules. So for (15)-(18) an alternative has to be given.

The advice given in section 3.4 stimulates us to ask just for what we need for generating *easy to please*. We need an expression like *easy* and some TV-phrase. Let us, following PARTEE 1977a, assume that we have a special category $\overline{\text{ADJ}}$ which contains *easy, tough* etc. The resulting expression *easy to please* will be of the category ADJ'. Then we are guided to the following rule:

S_{603}: If $\alpha \in P_{\overline{ADJ}}$ and $\beta \in P_{TV}$ then $F_{603}(\alpha,\beta) \in P_{ADJ'}$: where
$F_{603}(\alpha,\beta) = \alpha$ *to* β.

The translation of (this) *easy* must be such that it may be combined with an TV-translation in order to obtain an expression of the type of translations of adjectives. Then the translation rule reads

T_{603}: If α translates as α' and β as β' then $F_{603}(\alpha,\beta)$ translates into $\lambda x\alpha'(\lambda y\beta'(y,\lambda P[{}^{\vee}P(x)]))$.

Rule S_{603} makes it possible to generate the expressions containing *easy to please* we mentioned above. Unfortunately, Partee does not provide an explicit semantics for the source of all her constructions (sentence (15)) so we cannot compare it with the semantic consequences of S_{603}; but I expect that she will finally end up with something close to the result of T_{603}. Concerning the syntax, it is demonstrated that our requirement guides us to a much simpler treatment.

In section 3.5 and 3.6 we have considered two examples concerning the derived verb phrase rule. These examples do not cover all possible applications of the rule. But the treatment given here shows that in any case the two kinds of examples for which Partee has used the derived verb phrases rule can be dealt with in a better way by means of total rules.

5. SUBCATEGORIZATION AND RULE SCHEMES

5.1. Hyperrules

An argument for distinguishing categories (given for instance in chapter 1, section 1.3) is that certain groups of expressions behave (syntactically or semantically) differently than other groups of expressions. If for some rule it turns out that the rule can only be applied to a subset of the expressions of its input category, then this can be considered as an indication that the system of categories is to coarse. A method to avoid partial rules consists of refining the system of categories. In this section we will consider examples of this method, and present tools which are useful when it is employed.

There are several arguments for distinguishing among the category of nouns the groups of mass nouns and of count nouns. One of the differences between the expressions of these two groups is their behaviour with respect to determiners. Let us compare, as an example, the count noun *ring* with the mass noun *gold*. Well-formed are *a ring* and *every ring*, whereas ill-formed are *a gold*, and *every gold*. In larger expressions the same differences arise: well-formed are *a beautiful ring* and *every ring from China*, whereas ill-formed are *a beautiful gold* and *every gold from China*. These differences constitute an argument for introducing in the grammar the separate categories 'Mass Noun' and 'Count Noun'.

In many respects, however, mass nouns and count nouns behave analogously. Expressions of both categories can be combined with relative clauses and with adjectives. If we treat mass nouns and count nouns as being two independent categories, then the consequence is that the rules for relative clause formation and for adjective addition are duplicated. Thus the grammar will contain a lot of closely related rules. This effect will be multiplied if more categories are distinguished among the nouns. Therefore it is useful to have a tool for controlling this proliferation. Such a tool are rule schemes.

Rule schemes are not new in Montague grammars; recall the rule for relative clause formation given in chapter 4.

$S_{3,n}$: CN × S → CN
$F_{3,n}$: replace in α all occurrences of he_n by *him/she/it*, and of him_n by *him/her/it*, according to the gender of the first noun or term in β; concatenate (α, *such that*, β).

This cannot be considered as a rule because $F_{3,n}$ deals with occurrences of he_n, whereas this expression does not occur in the lexicon of the fragment: examples of relevant expressions of the fragment for this rule are he_1, he_2. So we have to consider $S_{3,n}$ as a rule scheme out which rules can be obtained.

This can done by replacing all occurrences of n in the scheme by some number. Thus $S_{3,n}$ stands for an infinite collection of actual rules.

In $S_{3,n}$ three characteristic features are illustrated of the kind of rule schemes that I will use. The first one is that a rule scheme differs from a real rule by the occurrence of a parameter. $S_{3,n}$ contains the parameter n, which stands for a number. Schemes may contain several occurrences of one or more parameters, and I will put no restrictions on where a parameter stands for. The second characteristic feature is that out of a scheme an actual rule can be formed by means of substituting the same expression for all occurrences of a parameter. If it is not required that all occurrences are replaced by the same expression then the occurrences of the parameter will be indexed (e.g. with $n_1,n_2,\ldots$), and then occurrences with different indices may be replaced by different expressions. The third feature is that a parameter may stand for a part of a (formally simple) symbol. The expression he_n is, formally spoken, a single generator in the syntactic algebra, but in the scheme given above it is treated as a compound symbol with *he* and n as *parts*. This does not change the role of he_1 in the algebra; it remains a simple generator. One should distinguish the formal position in the algebra, and the presentation of an infinite collection operators (or generators) by means of schemes.

A rule scheme involving nouns is the following.

$S_{604,n}$: Adj $\times$ *cm* Noun $\rightarrow$ *cm* Noun

$F_{604,n}$: Concatenate (α,β).

From this scheme two actual rules can be obtained. If *cm* is replaced by 'Count', then we obtain a rule which says that an adjective in combination with a Count Noun forms a new Count Noun. If *cm* is replaced by 'Mass', then we obtain a rule which says that an adjective in combination with a Mass Noun forms a new Mass Noun. This scheme exhibits again the feature that a compound symbol in the sense of the scheme, can be a single symbol in the algebraic sense. In the algebra 'Count Noun" is a category symbol, whereas in $S_{604,n}$ it is a compound with 'Count' and 'Noun' as individual parts. Notice that the above scheme contains two parameters: n and *cm*.

The new formal aspects introduced in this section are the use of compound category symbols and the possibility to use parameters for parts of these symbols. The practical impact of this is that partial rules can be avoided by increasing the number of categories, and that rule schemes can be used for handling these categories.

Now I will introduce some terminology. The parameters in the rule schemes are called *metavariables*. To distinguish the rule schemes of the kind just described, from others, the former are called *hyperrules* (i.e. they are rules containing metavariables). Hyperrules without the occurrence of a variable are considered as a special case; by means of an 'empty' substitution they become actual rules. I will give the hyperrules a 'name' which starts with an H, and its parameters will not be included in the name. So rule $S_{604,n}$ mentioned above will be called H_{604}. The distinction between Count Nouns and Mass Nouns is in linguistics called subcategorization. I will use this term with the following formal interpretation. A category C_1 is called a subcategory of a category C_2 if the carrier of sort C_1 is a subset of the carrier of sort C_2.

5.2. Metarules

Suppose that we have a hyperrule which contains some metavariable. In the example from section 4.1 concerning nouns, I explicitly listed the two possible substitutions. But often the situation will be more complex. There are arguments for distinguishing among the nouns many more subcategories, and we will meet examples were infinitely many substitutions are possible. Therefore it is useful to have a handsome tool for telling what the possible substitutions for a metavariable are. In the sequel we will use rewriting rules for this purpose. Besides the grammar consisting of hyperrules I will give a second grammar, called *metagrammar*. This grammar consists of a collection context-sensitive rewriting rules, and in these rules the metavariables of the grammar occur as auxiliary symbols. If we take some metavariable as start symbol, then the metagrammar determines a language: the set consisting of all strings which can be produced from the metavariable which was taken as start symbol. The possible substitutions for a metavariable in some hyperrule are all strings from the language generated by the metagrammar using that metavariable as starting symbol.

The benefit of using a metagrammar becomes especially clear in cases were there are several levels of subcategorization and crosslinks in the category system. As example I present the metagrammar for the subcategorization system given in CHOMSKY (1965, p.85); it is striking to observe that Chomsky used rewriting rules as well for the presentation of the subcategorization.

$$\begin{array}{rcl} \mathit{common} & \rightarrow & \mathit{sgn}\ \mathit{count} \\ \mathit{sgn} & \rightarrow & \begin{cases} + \\ - \end{cases} \\ -\ \mathit{count} & \rightarrow & \mathit{sgn}\ \text{Abstract CN} \\ \mathit{count} & \rightarrow & \begin{cases} -\ \text{Animate CN} \\ \mathit{anim} \end{cases} \\ \mathit{anim} & \rightarrow & \mathit{sgn}\ \text{Human CN} \end{array}$$

According to the convention for substitution, this metagrammar implicates that a hyperrule containing *anim* as metavariable represents two actual rules (for the subcategories + Human CN and -Human CN), and that a hyperrule containing *common* represents 5 actual rules.

A grammar designed in the way sketched above is a system with two levels in the grammar: the level of metarules and the level of the (hyper)rules. The conception of a grammar with two levels is due to Van Wijngaarden, and was developed for the formal description of the syntax of the programming language ALGOL 68 (see VAN WIJNGAARDEN 1975). He used these notions hyperrule and metarule with about the same meaning (for a linguistically oriented example see VAN WIJNGAARDEN 1970). The same terminology, although with a somewhat different meaning, is used in GAZDAR & SAG 1981 and GAZDAR 1982. The concept of a two-levelled grammar gives rise to an elegant method handling a lot of rules, even an infinite number. The method could easily be generalized to multi-level grammars. In Van Wijngaarden's original system the metarules have to be context-free, whereas I allowed for context sensitive rules. This liberty has no consequences since the generative power of system lies in the rules, and not in the metarules. In the example given above (Chomsky's subcategorization) the context sensitive rules turned out to be useful. If we would be more liberal, and allow to use a type-0 grammar as metagrammar instead of a context sensitive grammar, then this would have the consequence that the by the metagrammar produced language would be undecidable. Then it would not be decidable whether a substitution for a metavariable is allowed, and consequently the set of actual rules would not be recursive. Therefore type-0 grammars are in our framework not acceptable in the metagrammar.

5.3. Variables

The use of variables in a Montague grammar gives rise to certain problems. I will consider here two of them. A more extensive discussion will be given in chapter 8.

1. 'Left over'

   According to the PTQ rules we may generate the sentence *He$_3$ runs*. This is not a correct English sentence because it contains *he$_3$*, which is not a correct English word.

2. 'Not there'

   One might apply a rule which involves variables in a situation in which such variables are not there. In this way one obtains relative clauses, which do not contain a reflexive pronoun. An example is *the man such that Mary seeks a unicorn.*

In order to eliminate these two problems, in chapter 8 a restriction will be proposed that contains the following two conditions:

(I) The production of a sentence is only considered as completed if each syntactic variable has been removed by some syntactic rule.

(II) If a syntactic rule is used which contains instructions which have the effect of removing all occurrences of a certain variable from one of its arguments, then there indeed have to be such occurrences.

It is evident that requirement (II) can be guaranteed by means of a partial rule. To this aspect I will return later. Requirement (I) says that all stages of the derivation process have to meet a certain condition. So is appears to be a global filter. Since one can tell from the final result whether the condition is met, it reduces however to a final filter. As I explained in chapter V, filters are not acceptable in our framework. But the effect of (I) can be obtained by means of a partial rule as follows. Replace everywhere in the grammar the category of Sentences by the category of Protosentence (so the grammar produces Protosentences). Then we add an extra rule which produces a sentence out of a protosentence in case requirement (II) is fulfilled, and which is not applicable when this requirement is not fulfilled. Thus only sentences obying (I) are produced. Since I aim at avoiding partial rules, I have to provide an alternative method for the incorporation of the above two restrictions. This will be given below.

Categories are defined to be complex symbols consisting of two parts: a category name as we used before (e.g. S), and a representation of a set of integers. The set indicates which indices occur in the expressions of that (complex) category. So *he_2 or he_3* is an expressions of the category (T,{2,3}). Other examples are *He_4 runs* of the category (S,{4}), and *John* of the category (T,$\emptyset$). The language generated by the grammar is defined as the set of expressions of the category (S,$\emptyset$).

The hyperrules of the grammar contain variables for integers (n) and variables for sets ($set_1, set_2, \ldots$). The following notations are used.

$set_1 \ddot{\cup}\ set_2$	denotes the set obtained as union of the sets set_1 and set_2
set with n	is a compound expression indicating that *set* contains element n
set $- n$	is a compound expression denoting the set obtained by removing the element n from *set*.

The hyperrule corresponding with S_4 reads

H_4: $(T, set_1) \times (IV, set_2) \rightarrow (S, set_1 \cup set_2)$

F_4: replace the first verb in β by its third person present singular, concatenate (α,β).

This hyperrule states that set of the syntactic variables in the sentence is the union of the syntactic variables in the T-phrase and the IV-phrase. An example of an actual rule obtained from H_4 is

H_4: $(T,\{1,2\}) \times (IV,\emptyset) \rightarrow (S,\{1,2\})$

F_4: see above.

This rule may be used in the production of *He_1 or he_2 runs*. Corresponding with $S_2, S_5, \ldots, S_{13}$ and S_{17} we have analogous hyperrules. The hyperrules corresponding with the rules S_{14} and S_3 are:

H_{14}: $(T, set_1) \times (S, set_2\ with\ n) \rightarrow (S, set_1 \cup [set_2 - n])$

F_{14}: substitute (α, first occurrence of he_n in β);
replace all occurrences of he_n in β by *he/she/it* and of him_n by *him/her/it* according to the gender of the first noun or term in α.

H_3 : $(CN, set_1) \times (S, set_2\ with\ n) \rightarrow (CN, set_1 \cup [set_2 - n])$.

F_3 : Replace he_n in β by *he/she/it* and him_n by *him/her/it*, according to the gender of the first CN in α;
concatenate (α, *such that*, β).

An actual rule obtained from H_{14} is

H_3: $(T,\emptyset) \times (S,\{2,3\}) \rightarrow (S,\{3\})$.

An application of this rule is the production of *John loves him_3* from *John* and *he*$_2$ *loves him*$_3$.

A formalist might object to the hyperrules given above since they implicitly assume that the reader knows what sets are, and what is meant by the symbols $\cup$, *with* and $-$. This is, however, not knowledge about operations of the grammatical system, but set theoretical knowledge, and the rules should not be dependent on this knowledge. In appendix 3 of this book it will be shown how these notions can be described by means of purely grammatical tools (viz. by rewriting rules).

Clause (I) required that the expressions of the generated language do not contain any occurrences of syntactic variables. In my approach this requirement is not formalized as a filter or as a condition in a partial rule, but within the system of categories. This is theoretically more attractive, and practically somewhat simpler. Clause (II) requires that in case a rule is applied which removes variables, then there are such occurrences. This clause is also dealt with in the categorial system, as one can see from the following. Let us suppose that the categorial information given in the rules corresponds with the syntactic operations performed by these rules (i.e. if the rule removes all occurrences of a variable, its index is removed from the set mentioned in the category of the produced expression). This assumption can easily be checked from the rules. Assuming this correspondence, the condition *set*$_2$ *with* n in H_{14} and H_3 guarantee that these rules are applied only to expressions containing the required occurrences of variables. So instead of formalizing (1) as a condition in a partial rule, it is formalized within the categorial system. This is theoretically more attractive, but practically somewhat more complex.

One observes that the requirements concerning variables can be dealt with in accordance with the aim of using total rules. This is made manageable by using a two-level grammar. Within this system the requirements can be handled about as easy as in a system with partial rules. But the introduction of two levels did not make the system simpler. Therefore I would not say that the requirement of using total rules has led us here to a simpler treatment. In order to see practical advantages of using a two-level grammar, one has to consider a much more complicated situation. Such a situation will be described in chapter 9: the interaction of tense scope, and quantifier scope. But in the present situation the advantage is only of theoretical importancy. Therefore one might take in practice the following position. It has been shown that the requirements concerning variables can be incorporated within a system with only total rules. This implicates that in practical cases there is no need to treat the requirements explicitly in this way. One might use requirements (I) and (II) as they are formulated, assuming the present formalization.

5.4. A theoretical result

The method introduced in this section for eliminating partial rules consists in refining the system of categories. For nouns I gave an example with five subcategories, and for the treatment of variables even an infinite number. One might consider the possibility of applying this method up to the very limit (every expression constituting a single category on its own). By proceeding that far, all partial rules are eliminated from the grammar. This simple idea is followed in the proof of the following theorem (LANDSBERGEN 1981).

5.1. THEOREM. *For every enumerable algebraic grammar* G *with partial rules, there is a general algebraic grammar* G' *with total rules, such that* L(G) = L(G').

PROOF. We obtain G' as follows. For each category C of G and each expression w of this category, we define a new category in G', denoted by the compound symbol (C,w). The only expression of this category is w. Since for each sort of G, the expressions are recursively enumerable, the sorts of G' are recursively enumerable as well (but in general not recursive). For each rule R in G there is a collection of rules in G'. If according to a rule of G the expression w_0 (of category C_0) is formed out of the expressions $w_1,w_2,\dots,w_n$ of the categories $C_1,\dots,C_n$, then there is in G' a rule producing expressions of the category (C_0,w_0) out of expressions of the categories $(C_1,w_1)\dots(C_n,w_n)$. Of course, this rule can be used in only one production, but it is a total rule. Since the rules of G and the expressions of L(G) are recursively enumerable, the rules of G' are recursively enumerable as well. Suppose that the distinguished category of G is S (so $L(G) = G_S$). Then we add for each category (S,w), where w is arbitrary, a new rule which takes as input an expression of category (S,w) and yields as output the expression w of category S. From this construction it is evident that L(G) = L(G').
5.1. END

The theorem states that every language generated by a grammar with partial rules can be generated by a grammar with total rules. As such the theorem is not surprising: even finite grammars have a universal generating capacity. The merit of the theorem lays in the method used in its proof. The grammars G and G' do not only generate the same language, but they do so in the same way. The derivational history of a given expression has in G and in G' the same structure. Several properties of G are carried over to G'; for instance, if G consists of concatenation rules only (i.e. if the rules correspond with a context free rules), then the same holds for G'. This correspondence between G and G' means that the proof can be used for restricted classes of grammars as well.

One might be tempted to conclude from the theorem that grammars with partial rules are just notational variants of grammars with total rules, and that it constitutes a justification for writing partial rules in a framework that requires total rules. This is however not the case, since an important property of G can be lost by transforming it to G'. If G is a recursive grammar, where its generated language L(G) is not recursive, then G' is not a recursive grammar. In chapter 2 we have restricted our attention to the class of recursive grammars. Hence the method used in the theorem may bring us outside the class of grammars we are working with. For this class the grammars with partial rules cannot be considered as a notational variant of the grammars with total rules. So the requirement to use total rules is a substantial one. It has a consequence that not every condition on applicability is acceptable: only those are acceptable which can be reformulated as total rules in a recursive algebraic grammar. In previous sections it has been demonstrated that such a reformulation gives rise to a simpler, a better grammar.

6. THE WELL-FORMEDNESS CONSTRAINT

In this section I will discuss some aspects of a principle for syntax due to Partee. It is called 'the well-formedness constraint', and it reads as follows (PARTEE 1979b, p.276):

Each syntactic rule operates on well-formed expressions of specified categories to produce a well-formed expression of a specified category.

The motivation for this principle is related with the aim 'to pursue the linguists goal of defining as narrowly as possible the class of possible grammars of natural languages' (op. cit. p.276). Although this is a completely different aim than the theme of the present chapter, it turns out that the well-formedness constraint has practical consequences which can be compared with consequences of our algebraic approach, in particular with the requirement of using total rules. I will restrict the discussion of the well-formedness constrain to these aspects.

Our investigations started from algebraic considerations, and the requirement of using total rules constitutes a formal restriction. The value of the requirement was its impact on heuristics. What is the position of the well-formedness constraint in this respect? Is it a formal restriction, heuristic guideline, or something in between these two? I will first try to answer this question by considering the constraint as it is formulated; Partee's interpretation will be considered thereafter. In order to answer the question concerning the formal position of the well-formedness constraint, it is important to have a formal interpretation for the phrase 'well-formed expression'. I will consider two options.

One might decide to associate the phrase 'well-formed expression' with the meaning that this phrase has in formal language theory. The rules of a grammar produce strings over some alphabet, and these strings are called the well-formed expressions over this alphabet. The epithet 'well-formed' is used to distinguish these strings from the other strings over this alphabet. Un-well-formed generated expressions do not exist by definition. It is possible to tell what the well-formed formulas of predicate logic are, but it is not possible to give examples of un-well-formed formulas of predicate logic: if a string is not well-formed, it is no formula at all. If we apply this interpretation to the PTQ grammar, then we have to conclude that *love him*$_1$ is a well-formed expression (of the category IV) because it is a string produced by the grammar, whereas *love her* is not well-formed (because it is not produced as IV-Phrase). With this interpretation the phrase in the constraint stating that the rules produce well-formed expressions is a pleonasm. The same holds for the input: the only possible expressions of specified categories are the expressions generated by the grammar. With this interpretation the well-formedness constraint just describes how the framework operates, and it is no constraint at all.

One might relate the notion well-formedness with the language generated by the grammar. Then the generated language consists of well-formed expressions, and also all substrings of well-formed expressions are considered as well-formed. Following this interpretation, the constraint says that all intermediate stages in the production process have to be substrings of the produced language. So an acceptable grammar for English has not only to be adequate (i.e. produce the correct language), but also all the intermediate stages arising from the grammar have to be adequate in a certain sense. This mixture of the notions 'possible grammar' and 'adequate grammar' makes the constraint an unusable one. Suppose that a list of rules of a grammar for English is presented, and one is asked whether they conform the constraint. In order to answer this question one may start to produce some strings by means of the rules, and ask for each application of a rule, whether it is applied to well-formed expressions of English. Suppose that this is the case, then one cannot thereby conclude that all rules from the list obey the constraint, since not all possible derivations have been considered. One has to try and try again, but the definite answer 'yes' cannot be given. It may be undecidable whether an arbitrary grammar satisfies the constraint or not. Of course, this is not a mathematical proof. Such a proof cannot be provided, since the set of English sentences is not a mathematically defined set, but related questions in formal language theory are known to be recursively undecidable. Since the constraint is an undecidable constraint, it cannot be accepted as a restriction on the class of possible grammars (otherwise a more attractive, undecidable, constraint would be 'is an adequate grammar for English').

Partee gives no formal definition of the notion 'well-formed expression'. Conclusions about her interpretation have to be based upon the examples she gives concerning the constraint. As an illustration of the constraint she presents a rule which forms adnominal adjectives from relative clauses. (PARTEE 1979b,p.277). Its syntactic function F_i has the effect that:

F_i(*immigrant who is recent*) = *recent immigrant*.

The input for this operation is an ill-formed expression (*immigrant who is recent*), and therefore she judges that this rule is prohibited by the well-formedness constrain. From this example is clear that she does not follow the first interpretation given above, the second one is closer to her intentions. But she would not consider all substrings of well-formed expressions as being well-formed as well (Partee, personal communication). I expect that *John and Peter* is well-formed, whereas *John and* is not. Probably the judgement what well-formed expressions are, is to be based upon linguistic intuitions. In any case, Partee does not give a formal interpretation for the notion 'well-formed expression'. If this notion is not formally interpreted, then the constraint itself cannot be a formal restriction either. Furthermore, both our attempts to give a formal interpretation were not successful.

I conclude that the constraint has to be considered as a guideline for designing rules. As such it might be useful for its heuristic value, but it has not the position of a formal constraint on the possible kinds for grammars. As a guideline it is a very appealing one, since it aims at a natural way of production; in which no artificial expressions occur as intermediate forms. However, following this interpretation is not without problems. As HAUSSER (1978) remarks, the intuitions concerning the well-formedness of incomplete expressions are rather weak. Hausser gives as example *and about the seven dwarfs quickly*; well-formed or not? Furthermore, the well-formedness constraint does, even in clear cases, not guarantee that only natural production processes are obtained. Hausser gives as example an operation with the following effect.

F_n(*John kissed Mary*) = *Bill walks.*

This operator F_n is according to the well-formedness constraint an acceptable operator: an intuitively well-formed expression is transformed into well-formed expression. In order to give real content to the principle, restrictions have to be put on the possible effects of a rule. PARTEE (1979a) gives some proposals for such constraints, and her ideas will be followed in chapter 8.

A consequence of Partee's interpretation of the well-formedness constraint brings us back to the discussion of this chapter. Her interpretation says that in all stages of the production process only well-formed expressions are formed. So there is no need to filter out some of them. Neither there is a need to have obligatory rules which have in transformational grammars the task to transform ill-formed expressions into well-formed ones. So in a grammar satisfying the constraint obligatory rules and filters are not needed. Partee even goes further and interpretes the constraint in such a way that they are disallowed. As we found in section 1, such requirements are a direct consequence of the algebraic framework.

Partee's proposal deviates in an important aspect from our framework. Following linguistic practice, she allows for partial rules. As explained in the previous sections, I would not like to follow this idea and I would prefer to use total rules. Some effects of the well-formedness constraint can be dealt with by means of the requirement of using total rules, as will be shown below.

Suppose that in a grammar with total rules there is a rule S_i of which the syntactic operation F_i has the following effect.

F_i(*immigrant who is recent*) = *recent immigrant.*

So the rule operates on a common noun phrase which, according to rule $S_{3,n}$ must be constructed from the common noun *immigrant* and a sentence of the form *he*$_n$ *is recent.* This sentence has to come from the IV-phrase *be recent.* Since we require that the rules are total, we may also combine this IV-phrase with other term-phrases. So the sentence *John is recent* also is generated by the grammar, and this is not a correct sentence of English. This example suggests that an adequate grammar for English with total rules cannot contain a rule which generates *recent immigrant* in the way S_i does, because one cannot get rid of the phrase *be recent.* But an easy solution for the production of *recent immigrant* is available. Follow the advice given in section 4.3, and ask for what we need to produce this phrase. This advice suggests us to ask for an adjective (*recent*) and a noun phrase (*immigrant*). So the requirement of using total rules has the same practical impact here as the well-formedness constraint: it is a guideline for obtaining a non-artificial production process. (Note that I did not prove that it is impossible to have a rule like F_i in a system with total rules; I expect that a refined subcategorization might make this possible).

PARTEE (1979b) discusses certain aspects of the formation of (13).

(13) *Fewer of the women came to the party than of the men.*

Following BRESNAN (1973), this sentence is derived from the (ill-formed) sentence (14) by means of an operation called Comparative Ellipsis.

(14) *Fewer of the women came to the party than of the men came to the party.*

This is in its turn derived from (15) by Comparative Deletion.

(15) *Fewer of the women came to the party than x many of the men came to the party.*

As Partee says, the production of (13) is a difficult case for the well-formedness constraint since it uses the ill-formed source (14). Partee says: 'Unless further analysis of these constructions leads to a different kind of solution, they would seem to require the admissibility of ungrammatical intermediate stages. (Note that the derivations in question give semantically reasonable sources, so any reanalysis has a strong semantic as well as syntactic challenge to meet).' (PARTEE 1979b,p.303,304).

For our requirement of using total rules this production process is problematic as well. It is no problem that the rules of comparative deletion and comparative elipsis are partial rules, since they are meaning preserving. But the production of the ill-formed sentence (14) is problematic since we cannot get rid of this sentence: we cannot filter this sentence out, we may not have it as an expression of the generated language, and we may not use its embeddings (cf. the discussion concerning *recent immigrant*). But why follow this approach? Maybe one judges that a source like (14) or (15) expresses the semantic content of the comparatives more completely than comparatives. Or one wishes to explain the semantic relations between all variants of comparatives by generating them from the same kind of source. In transformational grammar this might be valid arguments, no other formal tools than transformations are available. In a Montague grammar there is a semantic component in which such semantic relations can be formally expressed. So if we do not need such a source for syntactic reasons we may try another approach. The requirement of using total rules guides us toward asking what we need. In order to make a sentence of which the kernel consists of two terms and a verb phrase, we need two terms and a verb phrase. Therefore we should introduce a three place rule

F_{605} *(John,Bill,see women)* = *John sees more women than Bill.*

The semantic component has to express what is compared; the syntax needs no to do so.

Another rule might compare of two nouns in which degree they are involved in a certain property.

F_{606} *(man,boy,come to the party)* = *fewer of the men come to the party than of the boys.*

One may also compare two terms for two verb phrases

F_{607} *(John,Bill, see men,meet women)* = *John sees more men than Bill meets women.*

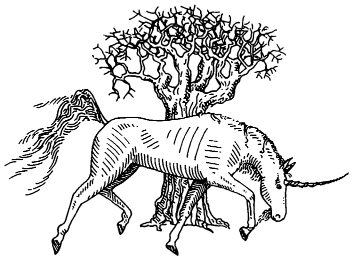
These examples do not provide for a treatment of the comparative. They just illustrate the kind of solution one might search for in a framework with total rules. Variants are possible: for instance, one might introduce compound quantifier phrases like *fewer of the man than of the boys,* and use instead of F_{606} a rule with two arguments. Note that all these attempts to find total rules, are in accordance with the well-formedness constraint.

CHAPTER VIII

CONSTITUENT STRUCTURES

ABSTRACT

Some proposals from the literature for assigning constituent structures to the expressions produced by a Montague grammar are shown to violate the framework. A treatment of the syntax of the PTQ fragment is presented which assigns constituent structures to the produced expressions and which meets the requirements of the framework. Furthermore a restricted set of syntactic operations is introduced for the description of the syntactic rules.



1. STRUCTURE - WHY?

The syntactic rules of PTQ make a primitive impression in comparison to the kind of rules used in transformational grammars. A first point of difference is that the syntactic operations make no reference to the constituent structure of the involved expressions. A second one is that the syntactic operations are described without any formalism: the desired effects are described by English sentences. On the one hand English is a rather poor tool since in this way the description of the syntactic operation can hardly use any abstract syntactic information. At the other hand it is a very unrestricted tool, since it allows any operation that can be described in the English language. Since the earliest times of Montague grammar, it has been tried to bring the syntax of Montague grammar closer to that of transformational grammar. This would open the possibility to incorporate syntactic knowledge from transformational grammars in Montague grammar, and to discuss the differences. In this chapter I will present the first steps of an approach which makes the syntax of Montague grammar less primitive: by developing a formalism for the formulation of the syntactic rules, and by introducing constituent structures in the syntax.

An example of the kind of structure used in transformational grammars is given in figure 1. The tree is not taken from any proposal in that field (then several details would be different), but it can be used to illustrate what kind of information is provided by such trees. The words attached to the end nodes of the tree yield, when read in the given order, the sentence of which the tree represents the constituent analysis. Constituents are groups of words which have a certain coherence. This appears for instance from the fact that it is rather easy to replace a constituent of a sentence by another group of words, whereas this is not the case for arbitrary groups of words from the sentence. The tree in figure 1 indicates what the constituents of the sentence are: all words of a certain constituent are connected to the same node in the tree. This node is labelled by a symbol: the name of the category of the constituent. Thus the tree gives the information that each word is a constituent, and that e.g. *a unicorn* is a constituent, whereas *seeks a* is not.

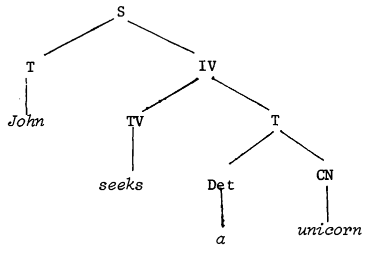


Figure 1 Tree like those in transformational grammar

A first argument for the introduction of constituent structures in the syntax of Montague grammars is that it would make it possible to incorporate ideas, or even particular rules, from transformational grammars into Montague grammar. I will not try to sketch the role such structures play in the syntax of transformational grammars; the reader should accept that constituent structures have proven their usefulness. A second argument is that, even without the aim of incorporating ideas from transformational grammar, it is useful to have structural information available about the expressions dealt with. An example, based upon a phenomenon from the PTQ grammar, is the following (PARTEE 1973).

Rule S_{11a} from PTQ, the rule for verb-phrase conjunction, produces the IV-phrase

(1) *walk and talk.*

Rule S_8 produces from (1) and the verb *try to* the IV-phrase

(2) *try to walk and talk.*

From the term *John* and the IV-phrase (2) we can produce, according to rule S_4, the sentence

(3) *John tries to walk and talk.*

Another derivation is to produce first (using S_8)

(4) *try to walk.*

Next we produce (5), using S_{11}.

(5) *try to walk and talk.*

Application of S_4 to (5) yields (3), but the correct form of a sentence with the intended conjunction would be

(6) *John tries to walk and talks.*

In order to correct rule S_4 for this, it is useful to distinguish the conjunction of *try to walk* and *talk* form the IV phrase *try to walk and talk.* So it is useful to assign structure to the strings (5) and (6).

This second argument shows that it is useful to have some kind of structure available, not that it has to be the kind of structures used in transformational grammars. As has been shown by FRIEDMAN (1979), the kind of problems mentioned above can be dealt with by un-labelled trees. A completely different kind of syntactic structures is used in JANSSEN 1981b, where the present framework is combined with the structures used in Dik's 'functional grammar' (DIK 1978,1980). However, the kind of structures I will consider in this chapter are constituent structures of the kind described above.

2. THEORETICAL ASPECTS

2.1. Trees in Montague grammar

The rules of a Montague grammar determine how basic syntactic units are combined to larger ones. Such production processes can be represented by a tree. The tree for the de-dicto reading of *John seeks a unicorn* is given in figure 2.

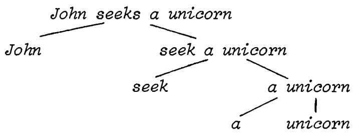


Figure 2 tree from Montague grammar

Such trees are representations of derivational histories. For this reason PARTEE(1975) compares them with the T-markers from transformational grammar, and not with the produced trees themselves. In transformational grammars trees are produced, and if one wishes to compare the approach of Montague grammar to the approach of transformational grammar, then one has to compare trees. Trees like the one in figure 2 are the only trees one finds in publications of Montague. Therefore one is tempted to compare such trees with the trees obtained in transformational grammars.

The tree in figure 2 is not of the form of the trees of transformational grammars. The main difference is that in transformational grammars the nodes are not labelled with expressions, but with category symbols (except for the end-nodes). Therefore one considers the tree from figure 2 as an unusual representation of the tree given in figure 1. Then the tree from figure 2 is taken as the syntactic structure assigned to the sentence by the PTQ grammar. Proceeding in this way, using the only trees available in Montague grammars, it becomes possible to compare the structures in Montague grammar with the structures in transformational grammars. This view on syntactic structure in Montague grammar can be found in work of several authors. In the next chapter we will see that PARTEE (1973) has compared the relative clause formation in Montague grammar and in transformational grammar by comparing trees like the one in figure 2, with those of transformational grammars. This way of discussion was followed up in by BACH & COOPER (1978). The same approach can be found in COOPER & PARSONS (1976). They describe a transformational grammar that is claimed to be equivalent with the PTQ system. The base rules of their transformational grammar produce (roughly) the same trees as the derivational histories of PTQ.

If one just compares the trees in the two approaches one soon will find great differences, and problems arise if one wishes to take the trees from Montague grammar as serious proposals for the syntactic structure assigned to a sentence. Consider the tree for the de-re reading of *John seeks a unicorn*, given in figure 3, or alternatively the one in figure 4.

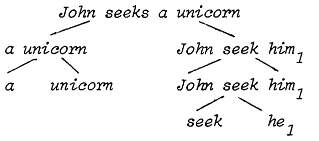


Figure 3: de-re reading

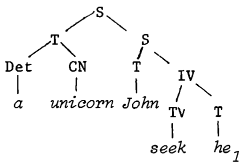


Figure 4: variant of figure 3

This tree cannot be taken as a serious analysis of the constituent structure of the sentence since it does not even fulfill the weakest requirement: that the lexical material is presented in the correct sequence.

Cooper has developed a variant of Montague grammar in which no quantification rules are used, and which seems to eliminate the problem just mentioned. I give an example from COOPER 1978 (the idea originates from COOPER 1975). Consider the tree in figure 2. The interpretation of this tree follows its structure. The lexical items are interpreted first, and next the interpretations of larger constituents are formed. The usual interpretation yields the de-dicto reading, roughly presented as *John'(seek'(a unicorn'))*. The de-re interpretation is obtained by means of a mechanism which gives the translation of *the unicorn* wide scope. This mechanism simply is a storage mechanism which allows the translation of the noun phrase *a unicorn* to be stored, putting a variable placeholder in the regular translation. The stored translation of a unicorn is carried up the tree, until it can be retrieved at a suitable point where quantifying in is allowed. The store is a set of pairs consisting of an interpretation of a term and a variable. The way of processing is as follows.

$$\begin{array}{rl} \textit{a unicorn} & \rightsquigarrow \langle \lambda P\,{}^{\vee}P(x_0),\ \langle \textit{a unicorn}', x_0\rangle\rangle \\ \textit{seek a unicorn} & \rightsquigarrow \langle \textit{seek}'(\lambda P\,{}^{\vee}P(x_0)),\ \langle \textit{a unicorn}', x_0\rangle\rangle \\ \textit{John seeks a unicorn} & \rightsquigarrow \langle \textit{John}'({}^{\wedge}\textit{seek}'(\lambda P\,{}^{\vee}P(x_0))),\ \langle \textit{a unicorn}', x_0\rangle\rangle \\ & \quad \text{retrieve from store, yielding} \\ & \langle \textit{a unicorn}'(\lambda x_0(\textit{John}'({}^{\wedge}\textit{seek}(\lambda P\,{}^{\vee}P(x_0))))),\emptyset\rangle. \end{array}$$

Cooper is not very explicit about the details of his proposal, and therefore it is difficult to evaluate it. Nevertheless, I have serious doubts about the acceptability of his proposal in any approach which accepts the principle of compositionality of meaning. The reason for this is as follows. The phrase *seek a unicorn* has two parts: *seek* and *a unicorn*. The contribution of the latter part to the meaning of the whole phrase consists in three components, one of them being the variable x_0. We have formalized meanings as abstract functions (intensions), and the symbol x_0 is not an element in this formalization. I assume that Cooper does not intend to define meanings as something which has the symbol x_0 as a component. So the mechanism does not build meanings from meanings, and therefore it violates the principle of compositionality of meaning. A more explicit description of a storage mechanism is given in PARTEE & BACH (1981); that proposal is discussed in LANDMAN & MOERDIJK (1983), where is shown that related objections apply.

The above discussion shows that we cannot get rid of trees like the one given in figure 3 by using Cooper storage. This has the following consequence. If one takes the tree representing the derivational history of a sentence in a Montague grammar to be the syntactic structure assigned to that sentence, then one has to conclude that in certain cases they are unacceptable as constituent structures. This is a practical reason against identifying the derivational histories with constitutent structures. As will be explained below, there are also algebraic reasons against it.

2.2. Algebraic considerations

In our framework the syntax is an algebra, i.e. a collection of carriers with operations defined on them. An algebra can be defined in many ways. For instance, one can enumerate all the elements of each carrier, and state what the operators are. But we have developed a more efficient way of defining an algebra: state what the generators and the operators are. In this way with each element of the algebra (at least) one derivational history can be associated. Such derivational histories are important for the semantics, because this process is mirrored when building the corresponding meanings. We have met several examples where the choice of a certain generated algebra was determined by semantic considerations. If we consider only the syntactic side of the situation, the generation process is just some method to define the algebra. If we would replace a given definition by another definition of the same algebra, the elements and the operators would remain the same. More in particular, an element of an algebra by itself does not have a derivational history. Only if one has additional information concerning the way in which the algebra is defined, it becomes possible to associate with an element some derivational history, and with the algebra itself an algebra of derivational histories (a term algebra). The operators of a syntactic algebra are functions defined on the elements of that algebra, and since the information how the algebra is defined, cannot be read off from these elements, the operators of the syntactic algebra cannot interfere with derivational histories. In section 2 I argued that we need syntactic structures in order to design more sophisticated rules. As argued above, the syntactic rules are completely independent of such histories. Hence we cannot consider derivational histories to be the structures we are looking for. This means that the only available trees cannot be used as a kind of syntactic structures. So the conclusion has to be that the PTQ grammar assigns *no structure at all* to the expressions it deals with.

If one wants the elements of an algebra to *have a structure*, then these elements *should be structures!* So in order to obtain a syntactic structure for the expression of a Montague grammar, this grammar should produce structures: trees, or, equivalently, labelled bracketings. This brings us to an approach dating from the first years of Montague grammar: PARTEE 1973. That proposal follows the sound approach to structure in Montague grammar. It distinguishes between the structure of the derivational history and the structure of the produced element itself. A remark about the relevance of distinguishing these two levels in a grammar for natural language can already be found in CURRY (1961), who calles the level of history 'tectogrammatics', and the level of produced expressions 'phenogrammatics'. DOWTY 1982 claims that rather different languages (such as Japanese and English) may have the same tectogrammatic structures, whereas the differences between the languages are due to phenogrammatical differences. This idea can also be found in LANDSBERGEN 1982, where it constitutes the basic idea for a computer program for automatic translation. In figure 5 the two kinds of structure are presented for the de-re reading of *John seeks a unicorn*: the trees within the squares are the constituent structures produced by the grammar, and the tree consisting of double lines with squares as nodes is the tree representing the derivational history.

2.3. Practical differences

Above I argued on algebraic grounds for distinguishing the structure an element has, from the derivational history assigned to it in some generative definition of the algebra. A practical aspect of this distinction is that there are completely different criteria for the design of these two kinds of structures. The derivational history is mapped homomorphically to the semantic algebra and determines the meaning of the expression. Semantic considerations play a decisive role in the design of the operators, and considerations concerning efficiency of definition determine the choice of the generators. The inherent constituent structure of the expressions is determined by syntactic considerations, e.g. the role such a structure has to play in the description of the syntactic effect of an operation. These

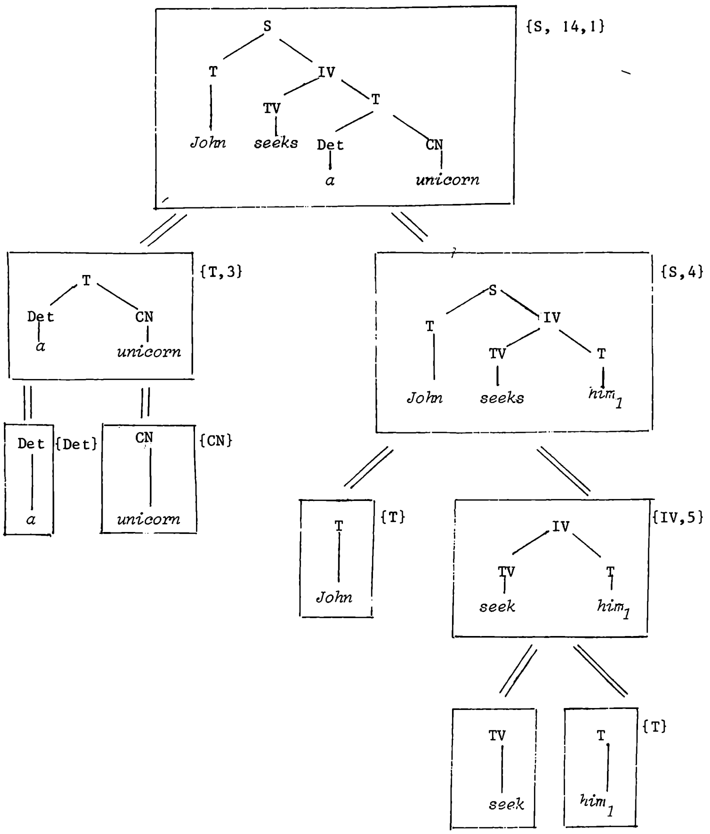


Figure 5 One derivational history containing many constituent structures

two different kinds of arguments may yield different kinds of structures. Below I will give some examples which show that the derivational history may sometimes differ considerably from what an acceptable constituent structure might be.

a) The PTQ rule S_{14} produces e.g. *John runs* out of *John* and *He*$_1$ *runs*. This is not an acceptable syntactic structure since it contains at an end node a word that does not occur in the sentence (cf. the discussion concerning figure 3).

b) In the grammar for questions by GROENENDIJK & STOKHOF (1981), there is a rule which substitutes a common noun into phrases of a certain kind. Thus *which man walks* is produced out of *man* and *which one walks*. Here the same argument applies as for the quantification rule of PTQ: it contains at an end node an argument that does not occur in the sentence.

c) In HAUSSER (1979b) another variant is presented of the substitution of a common noun for an occurrence of *one* in some phrase. Here the same conclusion holds.

d) BACH (1979a) presents rules which produce *persuade Bill to leave* out of *Bill* and *persuade to leave*. The operation which performs this task, called 'right-wrap', is a kind of substitution operation. It disturbs the sequence of words, and therefore it gives rise to a derivational history in which the order of the words does not correspond with the order of the words in the phrase. Therefore the derivational history is different from any possible syntactic structure.

e) DOWTY (1978) gives a very elegant categorial treatment of phenomena which are traditionally treated by transformations. Examples are dative movement and object deletion. His rules shift *serve* from the category DTV (takes a dative and a term), to the category TTV(takes two terms), and next to TV and IV. This history is presented in figure 6. As far as I know, such a structure has not been proposed in transformational grammars, which is an indication that there is no syntactic motivation for this structural analysis. All steps in this production process are semantically relevant, and I consider it as a prime example of a semantically motivated elegant design of a derivational history.

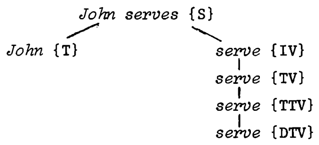


Figure 6: History à la Dowty

3. TECHNICAL ASPECTS

3.1. Introduction

In this section I will sketch some tools which are useful in a version of Montague grammar in which the syntax produces structured expressions. The desire to provide handsome tools for a certain limited purpose leads to restricted tools (all-purpose tools are usually not very handsome: I would not like to describe a language by means of a Turing machine). So, whereas I do not have the aim of Partee ('defining as narrowly as possible the class of possible grammars of natural languages' (PARTEE 1979b, p.276)), the practical work is closely related. The tools I will use originate mainly from Partee (ibid); in the details there are some differences. It is not my aim to develop a complete theory about structured syntax, but I will use the opportunity to make some theoretical and practical remarks about the available techniques. For more ambitious proposals which use the same approach to structure, see BACH 1979b, PARTEE 1979a, 1979b, and LANDMAN & MOERDIJK 1981.

The basic change I will make here in comparison with previous chapters, is a change of the algebra on the background, which is always assumed when we define a generated algebra. In the previous chapters this was mostly the algebra consisting of all strings over the alphabet involved with concatenation as operator. In the present chapter this background algebra is replaced by one which consists of all trees, labelled in an appropriate way, and which has the basic operations which will be described in the sequel.

3.2. Operations on trees

It is not very convenient to describe operations on trees by means of English sentences. Following Partee, I describe such operations as the composition of a few basic ones. These are described in the sequel.

The operation *root* gives a new common root to the members of a list of trees. The new root is labelled with a given category name. Let α and β denote trees, and let (α,β) denote the list consisting of these two trees. The effect of root((α,β),IV) is that the roots of the trees α and β are connected with a new root, labelled IV, see figure 7.

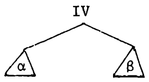
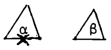
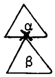


Figure 7: root ((α,β),IV)

The operation *insert* substitutes a tree for a given node in some other tree. Let us accept the phrase 'first he_2 in (α)' as a correct description of the node marked with x in tree α, see figure 8. Then the effect of *insert* (β, first he_n in (α)) is given in figure 9. A single word is considered as the denotation of a tree consisting of one node, labelled with that word. So the root operation can be applied to it. The effect of root(*and*, Con) is shown in figure 10.





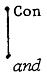


Figure 8: situation Figure 9: insert(α,β,x) Figure 10: root(*and*, Con)

These two operations for tree manipulation, together with operations for feature manipulation and index manipulation, suffice for the treatment of the PTQ fragment. For larger fragments other operations might be required. An example is 'everywhere-substitution', which has the effect of substitution for all occurrences of a variable. This effect cannot be obtained by means of a repetition of the *insert* operator since one and the same tree cannot be at the same time the daughter of different nodes. So everywhere-substitution requires a copy operation which might be added as a primitive operation. PARTEE 1979b has no copy operation, but considers everywhere-substitution as a basic operation (we do not need everywhere-substitution since we deal with $S_{14,n}$ by means of an operation on features).

As an example I present the rule for verb-phrase conjunction. If we stay close to PTQ, it gets the following form, and yields the result given in figure 11.

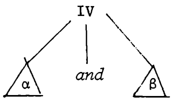
S_{11}: IV × IV → IV

F_{11}: root((α, *and*, β),IV).

One might prefer to give the connective *and* a categorial status in the syntactic structure; the status of a connective. Then the operation could read as follows, yielding the result given in figure 12.

S_{11}: IV × IV → IV

F_{11}: root((α,root(*and*, Con),β),IV)



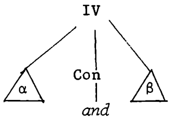


Figure 11: root((α, *and*, β),IV)   Figure 12: root((α,root(*and*, Con),β),IV)

3.3. Features and lexicon

Rule S_4 of PTQ tells that the subject-verb agreement in a sentence is obtained by replacing the first verb by its third person singular present. This is not an explicit formulation of what the effect of the rule is supposed to be. In an explicit form it would say that the verb *run* is to be replaced by *runs* and that *try to* (in PTQ a single word with a space inside) is to be replaced by *tries to*. Rule S_4 can have its short readable form only since it is not explicit about such details. In order to obtain an explicit syntactic rule which is not full with details, we have to abstract from the inflection behaviour of the individual verbs. So it is useful to let the syntactic rules deal with more abstract lexical elements. By incorporating features in Montague grammar, rule S_4 may for the PTQ fragment simply attach the features like *present* and *third person singular* to an elementary form of the verb without being concerned with the effect of these details for every individual verb. The information about morphological behaviour of the verb can be given in the lexicon or in a separate morphological component.

Features originate from transformational grammar. They were used, as far as I know, for the first time in Montague grammar by GROENENDIJK & STOKHOF 1976 for a phenomenon like the one above. Features are also useful if one incorporates transformations into Montague grammar. PARTEE 1979b gives several examples of transformations which require features; an example is the Subject-Aux inversion process for questions which requires isolation of the tense morpheme.

As an example of the use of features I give a reformulation of rule S_4 using features. Of course, the rule has in other respects the same shortcomings as the original PTQ rule, but it is fully explicit now.

S_4: $T \times IV \rightarrow S$

F_4: add features((*pres*,*sing* 3), first verb in (α));
root((α,β),S).

The details of the regular formation of the word forms can be given on a separate morphological component, whereas details about irregular word forms can be given in the lexicon. So the function *verbform* in the morphological component will be such that

verbform((*pres*,*sing*3),α) = α*s* (e.g.*walks*)
verbform((*pst*,*sing*3),α) = α*ed* (e.g.*walked*).

The morphological component also contains a function *pronomen* such that

pronomen(*sing*3,*acc*,*masc*) = *him*
pronomen(*sing*3,*nom*,*neut*) = *it*.

Besides morphological details, the lexicon also contains the information which features are inherent to the word (e.g. *John* always bears the feature *sing*3) and information about kinds of features for which the word may be specified (e.g. *John* may not be specified for tense).

On the basis on the above considerations I define a lexical element as a list consisting of the following five components.

1. a string being the basic form of the word
2. a category symbol, being the category of the lexical element
3. a list of inherent features
4. a list of kinds of features for which the lexical element can be specified.
5. a description of the procedure for making derived forms of the word.

The above definition says that a lexical element is denoted by a list of five elements, of which some are lists themselves. We already introduced a notation for lists. Let furthermore ( ) denote an empty list. The examples of lexical elements presented below are somewhat simplified with respect to PTQ since they only consider past and present tense.

("*John*",T, (*masc*,,*sing*3),(), wordform: "*john*")
("*walk*",IV,(),(*tense*,*pres*), wordform: *verbform*((*tense*,*pres*),"*walk*"),)
("*run*",IV,(),(*tense*,*pres*),

<u>if</u> value of tense = past <u>then</u> wordform: "*ran*"
<u>else</u> wordform: *verbform*((*pres*,*sing*3),"*run*")).

Up till now we only considered kinds of features which are well known. But nothing in the feature definition prohibits us to define unusual ones. We might define a feature kind 'mainverb' with values # and -. The instructions for introduction or deletion of this feature can be the same as the instructions for Bennetts # mark which indicates the main verbs of a phrase (BENNETT 1976). In this way we can use the features as syntactic markers. Following Partee, I would not like to do so. Features are introduced for isolating morphological phenomena, not for syntactic marking. So I would like to restrict features to morphological relevant ones, just as PARTEE (1979b) proposed. This restriction requires, however, a formal definition of this notion (Partee gives no definition).

The notion 'morphologically relevant feature' is clearly word dependent. The case feature is relevant for *he* but not for *John*. So we might call a feature morphologically relevant if it is relevant for at least some word in the grammar. But what does this notion mean? Something like that the feature influences the form of the word? It is to be required furthermore that this influence can be observed in real sentences: it is not enough that it occurs in some morphological rule since this leaves open the possibility of a fake feature which influences the form of some 'external' word that does not occur in a produced sentence. We want a feature to create an opposition of wordforms in the produced language. Based upon these considerations I would define the notion as follows.

3.1. DEFINITION. A feature F is called *morphologically relevant* in grammar G if the following two conditions are satisfied.

1. There is a sentence $S_1 \in L(G)$ containing a lexical element W which bears feature F and which has wordform W_1.
2. There is a sentence $S_2 \in L(G)$ containing an occurrence of W which does not bear feature F and which has wordform W_2 where W_2 is different from W_1.

3.1. END

Note that this definition uses quantification over the sentences in the language L(G). This quantification makes the notion 'morphologically relevant' to an undecidable notion. Suppose a list of syntactic rules, a lexicon containing features, and a list of morphological rules is given. Then one might try to show that a feature is not morphologically relevant by producing a lot of sentences and checking the conditions. However, one never reaches a stage that one can say for sure that such a feature is unacceptable. A formal proof is given in the following theorem.

3.2. THEOREM. *There exists no algorithm which decides for all grammars* G *and feature* F *whether* F *is morphologically relevant in* G.

PROOF. Suppose that such an algorithm would exist. Then this would give rise to a decision procedure for algebraic grammars, as will be shown below. Let G be an arbitrary algebraic grammar, with distinguished sort S, and suppose L(G) is a language over the alphabet A. Let α be an arbitrary string over this alphabet, and let $w \in A$ be the first symbol of α. Let $w' \in A$ be a new symbol, and F a new feature not occurring in G. Define the wordform of w when bearing feature F as being w'. Extend now grammar G to G' by adding the following rule:

R: $S \rightarrow S$ is defined by

$R(\alpha) = \alpha'$ where α' is obtained from α by attaching F to w

$R(\phi) = \phi$ if ϕ is not equal to α.

The only way to introduce w' in some expression of L(G') is by means of this new rule R. Hence F is morphologically relevant in G' if and only if $\alpha' \in L(G')$. From the definition of R it follows that $\alpha' \in L(G)$ iff $\alpha \in L(G)$. So if it would be decidable whether F is morphologically relevant, it would be decidable whether α is generated by grammar C. Since L(G) can be any recursively enumerable language, this question is undecidable.
3.2. END

The undecidability of the notion 'morphologically relevant' has as a consequence that it can not be considered as a formal constraint, and that it cannot be incorporated in the definition of the notion 'grammar'. This does not mean that the property is worthless. It could play about the same role as the well-formedness constraint, being an important practical guideline for designing and evaluating grammars.

3.4. Queries for information

In syntax one often uses information about the grammatical function of words and groups of words. The grammatical tradition has constituted names for most of these functions, e.g. mainverb, subject and object. That the information for determining these functions is present in the syntactic structure assigned to them, has already been stated in CHOMSKY 1965. He defines the subject of a sentence as the NP which is immediately dominated by the main S node. In spite of this approach to grammatical functions, the tradition of transformational grammar never uses such information explicitly. PARTEE 1979b proposes to incorporate this information in Montague grammar and to make explicit use of it in the syntactic rules.

On the question what the main verbs of a sentence are, an answer like *run* is not good enough since that verb might occur more than once. An answer has to consist of a list of occurrences of verbs; or formulated otherwise a list of nodes of the tree which are labelled with a verb. Functions used to obtain syntactic information such as mainverb are functions from trees to lists of nodes of that tree. The first use of functions of this kind in Montague grammar is given in FRIEDMAN 1979. Such functions are called queries by KLEIN (1979); PARTEE (1979b) uses the name properties for a related kind of functions. The different name covers a different approach to such functions. It is a property of each individual occurrence of a verb to be a mainverb or not so: hence a property is a boolean valued function, but a query is not. Since it is not convenient to use properties in syntactic rules, I use queries.

PARTEE (1979b) defines queries by means of rules parallel to the formation rules of the syntax. This has as a consequence that she in fact performs induction on the trees which represent derivational histories. Thus properties of derivational histories can be defined by means of her queries. It allows, for instance to define a query which tells us what the terms are which are introduced by means of a quantification rule. This query which I call 'substituted terms', can be defined as follows:

1. add to rule $S_{14,n}$ the clause
   substituted terms $(S_{14,n}(\alpha,\beta)) = \{\alpha\} \cup$ substituted terms (β)
2. do the same for the other quantification rules
3. add to the other rules the clause
   substituted terms $(S_i(\alpha,\beta)) =$ substituted terms$(\alpha) \cup$
   substituted terms(β)

Since we do not consider the derivational histories as representations of syntactic structures, we do not want information about the derivational history to be available in the syntax. Therefore I will not define queries in this way.

FRIEDMAN (1979) defines queries separately from the rules. She defines them for all constituent structures by means of a recursive definition with several clauses; each clause deals with a different configuration at the node under consideration. So Friedman performs induction on the constituent trees, and not on the derivational histories. Consequently, the query 'substituted terms' cannot be defined in Friedman's approach. In principle I will follow Friedman's method, but some modifications are useful.

Friedman's method has a minor practical disadvantage. If one adds a rule to the grammar, then at several places the grammar has to be changed: not only a rule is added, but all query definitions have to be adapted. In order to concentrate all these changes on one place in the grammar, I will mention the clauses of the definitions of a query within the operation which creates a new node, so within the operation root. In this way it is for each node determined how the answer to a query is built up from the answers to the query at its subnodes. If a root operation contains no specifications for a query, this is to be understood as that the answer to the query always consists of an empty list of nodes. As an example, I present rule S_{11a}, in which a clause of the query mainverb is defined.

S_{11}: IV × IV → IV

F_{11}: root((α,root(*and*, Con),β), IV,
mainverbs = mainverbs(α) ∪ mainverbs(β)).

The basis of the recursion for a query is formed by its application to a node only dominating an end node of the tree. Then a query yields as result that end node. So is the query mainverbs is applied to the root of the tree in figure 13, then the result is that occurrence of *run*.

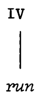


Figure 13: basis of recursion

4. PTQ SYNTAX

Below I will present a syntax for the PTQ fragment. The purpose of this section is to provide an explicit example of what a Montague grammar in which structures are used, might look like. It is not my aim to improve all syntactic shortcomings of PTQ; only some (concerning conjoined phrases) are corrected. For more ambitious proposals see PARTEE 1979b and BACH 1979.

In the formulation of the rules, several syntactic functions and operations will be used. Below the terminology for them is explained, thereafter they will be described.

1. *Queries*
Functions which have a tree as argument and yield a list of nodes in the tree. They are defined within the root operations.

2. *Primitive functions*
Functions of several types which yield information, but do not change anything.

3. *Primitive operation*
Operations of several types which perform some change of the tree or lists involved.

4. *Composed operations*
Like 3, but now built from other operations and functions.

QUERIES

Mainverbs

Yields a list of those occurrences of verbs in the tree which are the mainverbs of the construction.

Headnouns

Yields a list of those occurrences of nouns, pronouns and proper names which are the heads of the construction.

PRIMITIVE FUNCTIONS

Index of (w)

yields the index of the word w (provided that w is a variable)

First of (l)

yields the first element of list l.

All occurrences of (he_n,t)

yields a list of all occurrences of he_n in tree t.

Gender of (w)

yields the gender of word w, and of the first word of w if w is a list.

Is a variable (w)

determines whether term w is a variable (of the form he_n).

PRIMITIVE OPERATIONS

root((t_1,...,t_n),C, query:...).

Creates a new node which is the mother of the trees $t_1,\ldots,t_n$.
This new node is labelled with category symbol C.
For all queries a clause of their recursive definitions is determined: either explicitly, or implicitly (in case the query yields an empty list).

Add features (f,l)

Attaches to all elements of list l the features in feature list f.

Delete index (n,l)

Deletes index n from all elements in list l.

Replace index (l,m)

Replaces the index of all variables in list l by index m.

Insert (r,t)

Replaces node r by tree t, thus inserting a tree in another tree.

Union (l_1,l_2)

Yields one list, being the concatenation of lists l_1 and l_2.

COMPOSED OPERATIONS

<u>Termsubstitution</u> (t,n,r)

Substitutes term t in tree r for the first occurrence of he_n. The operation is defined by:

```
if is a variable (t)
then replace index (all occurrences of ($he_n$,r), index of (t))
else insert (t, first of (all occurrences of ($he_n$,r)))
    define: list=all occurrences of ($he_n$,r)
    delete index(list)
    add features ((sing,pers3,gender of (first of (main nouns (t))), list).
End definition.
```

Below the rules for the PTQ-fragment are given with exception of the rules for tense. Their formulation resembles the formulation they would get in an ALGOL 68 computer program I once thought of.

```
$S_2$: Det × CN → T
    root (($\alpha$,$\beta$),T,
    head nouns = head nouns ($\beta$)).

$S_{3,n}$: CN × S → CN
    define: list = all occurrences of ($he_n$,$\beta$);
    delete index (n, list);
    add features (gender of (head nouns ($\alpha$), list));
    root (($\alpha$, root (such-that,Rel),$\beta$),CN,
    head nouns = head nouns ($\alpha$)).

$S_4$  : T × IV → S
    add features ((pres,sing3), main verbs ($\beta$));
    add features (nom, head nouns ($\alpha$));
    root (($\alpha$,$\beta$),S,
    main verbs = main verbs ($\beta$)).

$S_5$  : TV × T → IV
    add features (acc, head nouns ($\beta$));
    root (($\alpha$,$\beta$),IV,
    main verbs = main verbs ($\alpha$)).

$S_6$  : Prep × T → IAV
    add features (acc, head nouns ($\beta$));
    root (($\alpha$,$\beta$), IAV).
```

S_7: IV/S × S → IV
root ((α,β), IV,
main verbs = main verbs (α)).

S_8: IV//IV × IV → IV
root ((α,β),IV,
main verbs = main verbs (α)).

S_9: S/S × S → S
root ((α,β), S,
main verbs = main verbs (β)).

S_{10}: IAV × IV → IV
root ((β,α), IV,
main verbs = main verbs (β))

S_{11a}: S × S → S
root ((α, root (*and*, Con),β), S,
main verbs = union (main verbs (α), main verbs (β))).

S_{11b}: S × S → S
root ((α, root (*or*, Con),β), S
main verbs = union (main verbs (α), main verbs (β))).

S_{12a} IV × IV → IV
root ((α, root (*and*, Con),β), IV,
main verbs = union (main verbs (α), main verbs (β))).

S_{12b}: IV × IV → IV
root ((α, root (*or*, Con),β), IV,
main verbs = union (main verbs (α), main verbs (β))).

S_{13} : T × T → T
root ((α, root (*or*, Con),β), T,
head noun = union (head nouns (α), head nouns (β))).

$S_{14,n}$: T × S → S
termsubstitution (α, n, β).

$S_{15,n}$: T × CN → CN
termsubstitution (α, n, β).

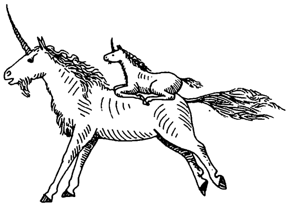
$S_{16,n}$: T × IV → IV
termsubstitution (α, n, β).

CHAPTER IX

RELATIVE CLAUSE FORMATION

ABSTRACT

Does the principle of compositionality compel us to a certain analysis of relative clause constructions? Answers given by Partee and Bach & Cooper will be investigated, and new arguments will be put forward. The question will be generalized and answered on the basis of algebraic properties of the framework. The investigations give rise to a restriction on the use of variables in Montague grammar: the variable principle.



1. INTRODUCTION

Our framework, which formalizes the principle of compositionality of meaning, says that the syntax and semantics are similar algebras, and that the meaning assignment function is a homomorphism. Now one may ask to what extent this organization of the grammar restricts the options we have in the syntax to describe a particular phenomenon. This question was raised by PARTEE (1973) with respect to relative clause constructions, and her answer was that we have to use a particular analysis. She concluded that the framework puts very strong constraints on the syntax, with the consequence that 'it is a serious open question whether natural language can be so described' (PARTEE 1973, p.55). Her argumentation is used by CHOMSKY (1975) to support his ideas of an autonomous syntax in transformational grammars. Partee's conclusion about relative clause formation has been disputed by BACH & COOPER (1978), who give an alternative construction.

In chapter 2 it has been proven that every recursively enumerable language can be described by means of a finite algebraic grammar. Hence Partee's question, as quoted above, has already been answered positively. But we will curtail it to the question whether the framework constrains the way in which natural language phenomena can be described. More in particular, we will investigate the thematic question: *does the framework of Montague grammar compel us to a particular syntactic analysis of restrictive relative clauses?* The arguments given in the literature will be considered, and new arguments will be put forward. In the course of the discussion positive and negative answers to the thematic question will alternate. An answer to the general version of the question is obtained as well. It will turn out that syntactic variables (like he_n) play an important role in relative clause constructions. This role is investigated, and this gives rise to the introduction of a new principle for Montague grammar: the variable principle. This chapter is a slightly revised version of JANSSEN (1981a).

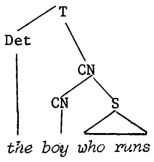
2. THE CN-S ANALYSIS

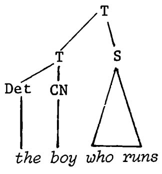
2.1. The discussion by Partee

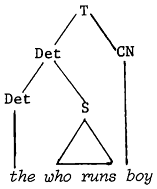
PARTEE (1973) considers three kinds of analyses of relative clause constructions which were proposed in the literature in the framework of transformational grammar. She investigates which of them constitutes a good basis for a compositional semantics. The comparison is carried out in the way described in chapter 7, section 2.1: the derivational histories from Montague grammar are compared with the constituent structures proposed in transformational grammars. As was explained there, this is not the most felicitous way to compare the two approaches. Our thematic question, however, does not concern a comparison but is a question about the present framework itself: the structures from transformational grammar merely constitute a starting point. Hence all trees under discussion have to be taken as representing derivational histories, even in case they originate from transformational grammar as constituent structures. In the sequel I will use the categorial terminology from the previous chapters, and not the transformational terminology used in the proposals under discussion.

Below I summarize Partee's argumentation. She discusses three kinds of analysis for the restrictive relative clause construction. They are named after the configuration in which the relative clause is introduced. These analyses (of which the second was the most popular among transformational grammarians) are

1. CN-S : the Common Noun-Sentence analysis (Figure 1)
2. T-S : the Term-Sentence analysis (Figure 2)
2. Det-S: the Determiner-Sentence analysis (Figure 3).







<u>Figure 1</u>: CN-S   <u>Figure 2</u>: T-S   <u>Figure 3</u>: Det-S

In the analysis presented in Figure 1, the common noun *boy* can be interpreted as expressing the property of being a boy, and the phrase *who runs* as expressing the property of running. The conjunction of these properties is expressed by the noun phrase *boy who runs*. The determiner *the* expresses that there is one and only one individual which has these two properties. So the CN-S analysis provides a good basis for obtaining the desired meaning in a compositional way.

In the T-S analysis as presented in Figure 2, the term *the boy* is interpreted as expressing that there is one and only one individual with the property of being a boy. Then the information that the individual is running can only be additional. So in a compositional approach to semantics *who runs* has to be a non-restrictive relative clause. Therefore Partee's conclusion is that the T-S analysis does not provide a good basis for a compositional semantics of restrictive relative clauses.

The Det-S analysis from Figure 3 does not provide a good basis either. The phrase dominated by the uppermost Det-node (i.e. *the who runs*), expresses that there is one and only one individual withthe property of running, and the information that this individual is a boy, can only be additional.

Of course, these arguments do not constitute a proof that it is impossible to obtain the desired meanings from the T-S and Det-S analyses. It is, in general, very difficult to prove that a given approach is not possible, because it is unlikely that one can be sure that all variants of a certain approach have been considered. This is noted by Partee when she says: 'I realize that negative arguments such as given against analyses 2. and 3 can never be fully conclusive. [...]' (PARTEE 1973, p.74 - numbers adapted T.J.). She proceeds: 'The argument against 3. is weaker than that against 2., since only in 2 the intermediate constituent is called a T.' (ibid.). Her carefully formulated conclusion is 'that a structure like 1, can provide a direct basis for the semantic interpretation in a way that 2 and 3 cannot' (ibid. p.54).

2.2. The PTQ-rules

Accepting the argumentation given in Section 2.1, is not sufficient to accept the claim that one should use the CN-S analysis. It remains to be shown that such an analysis is indeed possible, and this means providing explicit syntactic and semantic rules. Partee does not need to do so because in her discussion she assumes the rules for relative clause formation which are given in PTQ. Although these rules do not produce literarely the same string as she discusses, the same argumentation applies to them.

I recall the rule for relative clause formation given in chapter 4.

$S_{3,n}$: CN × S → CN
$F_{3,n}$: Replace he_n in β by *he/she/it* and him_n by *him/her/it*, according to the gender of the first CN in α; concatenate (α, *such that*, β)
$T_{3,n}$: $\lambda x_n[\alpha'(x_n) \wedge \beta']$.

According to this rule, the derivational history of *boy who runs* has the structure presented in figure 1. The phrase can be produced from *the* noun *boy* and the sentence *he*$_3$ *runs* by an application of instance $S_{3,3}$ of the above scheme. The corresponding translation reads

(1) $\lambda x_3[boy(x_3) \wedge run(x_3)]$.

This expression is interpreted as the property which holds for an individual if it both is a boy and is running. This is completely in accordance with the interpretation sketched for figure 1.

I recall that $S_{3,n}$ can be applied two times in succession (or even more). Then sentences are obtained like (2) (due to Bresnan, see PARTEE 1975, p.263) and (3) (due to PARTEE - ibid).

(2) *Every girl who attended a women's college who made a large donation to it was included in the list.*

(3) *Every man who has lost a pen who does not find it will walk slowly.*

In these sentences two relative clauses are attached to a single head noun. This construction is known under the name stacking (of relative clauses). In Dutch and German stacking is not a grammatical construction.

Rules $S_{3,n}$ and $T_{3,n}$ do not give a correct treatment of all phenomena which arise in connection with relative clauses. Some examples are:

1. The rule produces the *such-that* form of relative clauses, and this is not their standard form. A rule which produces a form with relative pronouns cannot be obtained by means of a straightforward reformulation of $S_{3,n}$, since complications arise (see RODMAN 1976).
2. In certain circumstances $T_{3,n}$ may give rise to an, unintended, collision of variables. This problem was discussed in section 5.3 of chapter 6; see also section 6.1.
3. Some famous problematic sentences do not get a proper treatment with this rule. Examples are the so called 'Bach-Peters sentences' and the 'Donkey sentences'. There are several proposals for dealing with them. For instance HAUSSER (1979c) presents a treatment for the Bach-Peters sentence (4), and COOPER (1979) for the donkey sentence (5).

(4) *The man who deserves it gets the price he wants.*

(5) *Every man who owns a donkey beats it.*

For a large class of sentences, however, the PTQ rule yields correct results, and I will restrict the discussion to this class. The class contains the relative clause constructions in the *such-that* form, the relative clause is a single (i.e. unconjoined) sentence, and stacking is allowed. Bach-Peters sentences and Donkey sentences are not included. For this class, the CN-S analysis gives a correct treatment in a compositional way, whereas for the T-S and Det-S analyses it is argued that this is not the case. So in this stage of our investigations, the answer to the thematic question has to be positive: the compositionality principle compels us to a certain analysis of relative clause constructions.

2.3. Fundamental problems

The PTQ rule for relative clause formation is essentially based on the use of variables in the syntax (he_n), and the use of unbound variables in the logic (x_n). This device gives rise to two problems which are of a more fundamental nature than the problems mentioned in Section 2.2. The latter concerned phenomena which were not described correctly by the given rule, but it is thinkable that some ingenious reformulation might deal with them. The fundamental problems I have in mind are problems which arise from the use of variables as such. It is essential for the entire approach to obtain a solution for these problems, since in case they are not solved satisfactorily we cannot use the tool at all. This aspect distinguishes them from the problems mentioned in Section 2.2. The problems also arise in connection with other rules dealing with variables ($S_{14,n},\ldots,S_{17,n}$). Note that the epithet 'fundamental' is not used to make a suggestion about the degree of difficulty of the problem, but to indicate the importance that some answer is given to it. The two fundamental problems are the following.

1) 'left-over'

The first problem is: what happens in case a variable is introduced that is never dealt with by $S_{3,n}$ or any other rule. On the syntactic side it means that we may end up with a sentence like *he*$_7$ *runs*. Since *he*$_7$ is not an English word, this is not a well-formed sentence, and something has to be done about it. On the semantic side it means that we may end up with an expression containing an unbound logical variable. From the discussion in Section 5 it will appear that it is not obvious how we should interpret the formulas thus obtained.

2) 'not-there'

The second problem is: what happens when a rule involving variables with a given index is applied in case such variables are not there. I give two examples of such situations. The first is obtained if one applies $S_{3,1}$ to the common noun *man*, and the sentence *Mary talks*. Then the noun-phrase (6) is produced, which is ill-formed because there is no pronoun which is relativized.

(6) *man such that Mary talks.*

On the semantic side (6) gives rise to a lambda operator which does not bind a variable. The second example (GROENENDIJK & STOKHOF 1976b) is obtained by an application of $S_{3,1}$ to *man* and *he*$_2$ *walks*. Then the common noun phrase (7) is formed, out of which (8) can be obtained.

(7) *man such that he*$_2$ *walks.*

(8) *He*$_2$ *loves the man such that he*$_2$ *walks.*

By an application of $S_{14,2}$ we finally obtain

(9) *John loves the man such that he walks.*

This sentence has just one reading, viz. that John loves a walking man. The translation rules of PTQ however, yield (10) as reduced translation for (9).

(10) $\exists u[\forall v[[man_*(v) \wedge walk_*(john)] \leftrightarrow u = v] \wedge love_*(john,u)]$.

This formula expresses that the one who walks is John. THOMASON (1976) makes a related observation by counting the number of ambiguities of (11).

(11) *Bill tells his father that John resembles a man such that he shaves him.*

For the first problem it is evident that it is the use of variables which creates it, and that it are not the phenomena themselves: if there were no variables in the syntax, they could not be 'left-over', nor remain 'unbound' in their translation. For the second problem it is rather a matter of conviction that it is the use of variables that creates the problem. Even if (6) would be well-formed, I would consider its production in the way sketched above, as an undesirable side effect of the use of variables, because it does not exhibit a phenomenon for which variables are required.

In the literature there are some proposals for dealing with these two fundamental problems. One proposal (implicitly given in RODMAN 1976) is of a purely syntactic nature and simply says: the 'left-over' and 'not-there' constructions are not acceptable, and in case such a construction threatens to arise, it is filtered out. This approach is not considered here in detail, because it played no role in the discussion concerning our thematic question. In the approach of COOPER (1975) the 'left-over' constructions are accepted, an answer is given to the semantic questions, and the 'not-there' constructions are dealt with in the semantics. In the next sections his proposal will be discussed in detail. A proposal combining syntactic and semantic aspects (JANSSEN 1980b) will be considered in Section 5.

3. THE T-S ANALYSIS

3.1. Cooper on Hittite

COOPER (1975) considers the construction in Hittite which corresponds to the relative clause construction in English. In Hittite the relative clause is a sentence which is adjoined to the left or the right of the main sentence. For this and other reasons, Cooper wishes to obtain such constructions by first producing two sentences and then concatenating them. A simplified example is the Hittite sentence which might be translated as (12), and has surface realization (13). The sentence is produced with the structure given in figure 4. For ease of discussion English lexical items are used instead of Hittite ones. 'Genitive' is abbreviated as 'gen', 'plural' as 'pl', 'particle' as 'ptc', and 'which' as 'wh'. The example is taken from BACH & COOPER (1978) (here and in the sequel category names are adapted).

(12) *And every hearth which is made of stones costs 1 shekel.*

(13) *SA*	*NA4*	*HI.A-ia*	*kuies*	*GUNNI.MES*	*nu*	*kuissa*	*1 GIN*
gen.stone-pl.-and			which	hearth-pl.	ptc.	each(one)	1 shekel

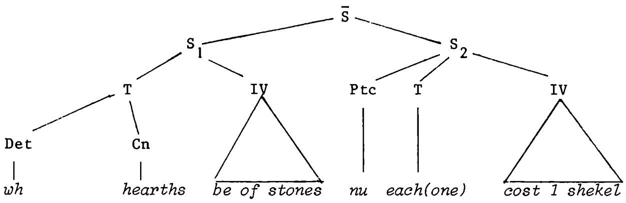


Figure 4

Sentence (13) is assumed to have the same meaning as the corresponding English sentence (12). There seems to be a conflict between the arguments in favor of a CN-S analysis as given in section 2, and the wish to use the S-S analysis for Hittite. Cooper's solution is to allow the Term-phrase *each(one)* 'to denote the set of properties possessed by every entity having property R' (BACH & COOPER 1978, p.147). Which property R is, is specified by the relative clause S_1. The translations of S_2 and S_1 are (14) and (15), respectively (here and in the sequel $^\vee$,$^\wedge$ and $_*$ symbols are added).

(14) $\forall x[{}^\vee R(x) \rightarrow \mathit{Cost\text{-}one\text{-}shekel}(x)]$

(15) $\mathit{Hearth}(z) \wedge \mathit{Made\text{-}of\text{-}stone}(z)$.

The syntactic rule which combines S_1 and S_2 to a phrase of the category $\bar{S}$, has as corresponding translation rule

$$\lambda R[S_2{}'](^\wedge\lambda z[S_1{}']).$$

Here $S_1{}'$ and $S_2{}'$ are the translations of S_1 and S_2, respectively. When this rule is applied to (14) and (15), we obtain (16) as reduced translation.

(16) $\forall x[\mathit{hearth}(x) \wedge \mathit{made\text{-}of\text{-}stone}(x) \rightarrow \mathit{cost\text{-}one\text{-}shekel}(x)]$.

Since $\bar{S}$ is of another category than S_1 and S_2, this production process does not allow for stacking, what is claimed to be correct for Hittite.

3.2. Bach & Cooper on English

BACH & COOPER (1978) argue that the treatment of COOPER (1975) of Hittite relative clauses can be used to obtain a T-S analysis for English relative clause constructions which is consistent with the compositionality principle. Terms are treated analogously to (the Hittite version of) *each (one)*. The term *every man* is assumed to denote, in addition to the PTQ interpretation, the set of properties possessed by *every man* which has the property R. Then the term-phrase *every man who loves Mary* is obtained from the structure given in figure 5.

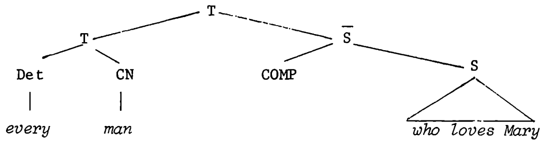


Figure 5

The rule for combining the translation of the term and the relative clause is:

$$\lambda R[\mathrm{T}'](^{\wedge}\mathrm{S}').$$

Here T' and S' are the translations of the term phrase and the relative clause, respectively. If we take (17) as translation of *every man*, and (18) as translation of the relative clause $\bar{\mathrm{S}}$, then we obtain (19) as translation of the whole term (after reduction).

(17) $\lambda P[\forall x[man(x) \wedge {}^{\vee}R(x)] \rightarrow {}^{\vee}P(x)]$

(18) $\lambda z[love_{*}({}^{\vee}z, mary)]$

(19) $\lambda P[\forall x[man(x) \wedge love_{*}({}^{\vee}x, mary)] \rightarrow {}^{\vee}P(x)]$.

Thus a T-S analysis is obtained for relative clause constructions, of which the translation is equivalent to the translation in the case of a CN-S analysis.

As Bach and Cooper notice, if we follow this approach, a complication has to be solved, since English allows for indefinite stacking of relative clauses. The proposal sketched so far, provides for one relative clause for each T. The complication can be taken care of by allowing an alternative interpretation not only for Terms, but also for relative clauses. 'Thus, for example, the relative clause *who loves Mary* can denote not only the property of loving Mary but also the property of loving Mary and having property *R*' (BACH & COOPER 1978, p.149).

Bach and Cooper remark that their compositional treatment of the T-S analysis clearly is less elegant and simple than the alternative CN-S analysis. They conclude: 'Our results seem to indicate, however, that such an analysis cannot be ruled out in principle, since any constraint on the theory that would exclude the T-S analysis, would seem to exclude the Hittite analysis as well. [...] or the happy discovery of some as yet unknown principles will allow the one, but not other.' (ibid. p.149).

The conclusion which prompts itself in this stage of our investigations is that the answer to the thematic question is a negative one: the principle of compositionality does not compel us to a special analysis of English relative clauses.

3.3. Fundamental problems

As a matter of fact, the discussion in BACH & COOPER (1978) does not provide the evidence that a T-S analysis is indeed possible for English relative clauses. They do not present explicit rules, and neither is it immediately clear what the details would look like (e.g. what is the role of $\bar{S}$ and COMP in the system of categories, and what is the translation rule which combines the translations of $\bar{S}$ and COMP). Nevertheless, the main point of their approach has become clear from their exposition.

The kernel of the approach of Bach and Cooper is to let the translations of terms and relative clauses contain a free variable R. For this variable the translation of some relative clause will be substituted. However, this variable R gives rise to the same kind of problems as mentioned in section 1 with respect to the variables x_n.

1. 'Left-over'
We may select for a term the translation with free variable R, whereas we do not use in the remainder of the production a rule which deals with this variable. Since R has no syntactic counterpart, the produced sentences are not per se ill-formed, but the question concerning the interpretation of unbound variables remains to be answered.

2. 'Not-there'
There may be an occurrence of the term-phrase *every man* with the translation without R, nevertheless appearing in a structure where a relative clause is attached to it. Then an incorrect meaning is obtained.

Only when these fundamental problems are solved, we may hope that the idea of Bach and Cooper leads to rules for the T-S analysis. Notice that the proposal of RODMAN (1976) for solving the two fundamental problems by filtering them out, cannot be followed here because in the syntactic expressions there is no variable which may control the filter. A solution has to be found on the semantic side. These problems for the Bach-Cooper idea, are signalized for the case of Hittite by COOPER (1975). He has proposed some solutions which are assumed by Bach and Cooper. In order to obtain further justification for the answer to the thematic question given in Section 3.2, we have to check the details of Cooper's proposals for these problems. This will be done in the next section.

4. THE PROPOSALS OF COOPER

4.1. Not-there

A translation rule which usually binds a certain variable, may be used in a situation where no occurrences of such a variable are present. To avoid problems, Cooper proposes to give no semantic interpretation to expressions of intensional logic which contain a vacuous abstraction. According to his proposal the interpretation of $\lambda R\alpha$ is undefined in case α has no occurrences of R.

Let us first consider in which way this idea might be formalized. At first glance it seems easy to obtain the desired effect. One just has to look into the expression α in order to decide whether $\lambda R\alpha$ is defined or not. However, this is not acceptable. Such an approach would disturb the homomorphic interpreation of intensional logic: for each construction of the logical language there is a corresponding interpretation instruction. To obtain the interpretation of a compound logical expression, the interpretations of the parts of that compound are relevant, but not their actual forms. An important consequence of this is that two semantically equivalent expressions are interchangeable in all contexts. If we would have a condition like 'look into α' in the definition of interpretation, this basic property of logic would no longer be valid. Two IL-expressions α and β might be semantically equivalent, whereas α satisfies the 'look into'-condition, and β not. Consequently, the interpretation of just one of $\lambda R\alpha$ and $\lambda R\beta$ would be defined. Such a violation of the fundamental law of substitution of equivalents is of course not acceptable, and therefore, a 'look into' clause has to be rejected. One has to respect the homomorphic interpretation of logic, and therefore, the situations in which $\lambda R\alpha$ should receive no interpretation have to be characterized in terms of the semantic properties of α (i.e. in terms of the interpretation of α with respect to a point of reference and a variable assignment). Cooper follows this strategy.

Cooper's first step towards a characterization consists of adding a restriction to the usual definition of the interpretation of $\lambda u\alpha$.
'[..] the function denoted by the abstraction expression $\lambda u\alpha$ is only defined for entities within its domain if a different assignment to the variable u will yield a different denotation for α' (COOPER 1975, p.246). As he notes, this definition has as a consequence that $\lambda u\alpha$ is 'undefined' not only if α does not contain a free occurrence of u, but also if α is

a tautology. Thus for instance, according to this definition $\lambda u[u=u]$ represents a function which is undefined for any entity. However, the technique of supervaluation [...] will show these expressions to be defined but not those where α is not a tautology' (ibid.). This definition is Cooper's final one, but it is not the one we need. It implies that now $\lambda R[x=x]$ is defined. This has the following consequence for relative clause formation. One might produce some sentence expressing a tautology, while its translation does not contain an occurrence of the variable R. Syntactically there needs not, in Cooper's approach, be anything which can prevent us from using this sentence in a relative clause construction, whereas, contrary to his intention, the interpretation of the translation is defined. So Cooper's definition does not provide a solution to the 'not-there' problem.

Cooper's aim was to give a semantic characterization of the IL-syntactic property 'contains an occurrence of the variable R'. I expect that there is no semantic property coinciding with the syntactic one. This is suggested by the observation that almost always a semantic irrelevant occurrence of a certain variable can be added to a given IL-expression. (ϕ and $R=R \wedge \phi$ are semantically indiscernable). Therefore, I expect that no solution in this direction can be found. Moreover, I consider the whole idea underlying Cooper's approach to be unsound. The standard interpretation of $\lambda R\alpha$ is, in case α does not contain an occurrence of R, a function that delivers for any argument of the right type, the interpretation of α as value. So $\lambda R\alpha$ denotes a constant function. Following Cooper's idea, one would loose this part of the expressive power of IL, a consequence I consider to be undesirable.

4.2. Left-over, proposal 1

The translation of a completed syntactic production of a sentence may contain an occurrence of a free variable. The second fundamental problem was what to do with variables that are 'left over'. Cooper proposes to assign no interpretation to such an expression, and to follow this approach for special variables only. Let z be such a variable (of the type of individuals). As was the case with the first problem, discussed in Section 4.1, one has to respect the homomorphic interpretation of IL. The desired effect should not be obtained by looking into the formula, but by changing the definition of interpretation. Cooper claims that the desired effect is obtained 'by restricting the assignments to variables so that z is always assigned some particular non-entity for which no predicate is defined' (COOPER 1975, p.257). This proposal gives rise to a considerable deviation from the model for IL as it is defined in PTQ. In that model, there are for every entity predicates which hold for it, e.g. the predicate of being equal to itself (viz. $\lambda u[u=u]$). This property is lost in Cooper's approach. He does not define a model which has the desired properties, nor does he give other details. For the discussion concerning the thematic question, this point is not that relevant, because BACH & COOPER (1978) do not propose to follow this proposal in the case of English relative clause constructions, but another one, which will be discussed in Section 4.3.

4.3. Left-over, Proposal 2

A second proposal of COOPER (1975) for the treatment of unbound variables which occur in the translation of a completed production of a sentence is to let the unbound variables be interpreted by the variable assignment function, and to give some linguistic explanation of how to understand the results thus obtained. This approach assumes that in complete sentences indices of variables can be neglected, or that there is some final 'cleaning-up' rule which deletes the indices. For our discussion of relative clause formation the syntactic details of this proposal are irrelevant because the variable R leaves no trace in the syntax.

The unbound relative clause variable R only occurs in subexpressions of the form $R(x)$. These subexpressions are understood by Cooper as 'a way of representing pragmatic limitations on the scope of the quantifier [binding x].[...]. Thus assigning a value to R in this case has the same effect as adding an unexpressed relative clause to show which particular set we are quantifying over' (COOPER 1975, p.258-259). The same strategy is employed in COOPER (1979a,b) for indexed pronouns. A pronoun he_n that has not been dealt with by a relative clause formation rule or some other rule, is considered as a personal pronoun referring to some contextually determined individual. Its translation has introduced a variable x_n, which remains unbound, and is interpreted by the variable assignment. This idea for dealing with free variables is also employed in GROENENDIJK & STOKHOF (1976b). In one respect the idea leads to a deviation from PTQ. There, an expression of type t is defined to be true in case it denotes 1 for every variable assignment (MONTAGUE 1973, p.259). So, $run(x)$ would mean the same as its universal closure. In the proposal under discussion this definition has to be dropped, but this does not cause any difficulties.

I have several objections against this proposal of Cooper. The first one is that it yields incorrect results; the other three argue that the whole approach is unsound. My objections are explained below.

1. If the translation of a phrase contains two occurrence of *R*, and a relative clause is combined with that phrase, then the translation of the relative clause is, by λ-conversion, substituted for both occurrences of *R*. As Cooper mentions, this phenomenon arises in his grammar for Hittite for (the Hittite variant of):

(20) *That(one) adorns that(one).*

Here the translation of both occurrences of *that(one)* contains an occurrence of the variable *R*. If this sentence is combined with a sentence containing two occurrences of a *wh*-phrase, semantically strange things happen. Cooper notes this problem and he says: 'My intuition is, however, that if there were such sentences, they would not receive the interpretation assigned in this fragment. [...] As it is not clear to me what exactly the facts of Hittite are here I shall make no suggestions for improving the strange predictions of the fragment as it is.' (COOPER 1975, p.260).

Unfortunately, the proposal for English of BACH & COOPER (1978) runs into a related problem. Consider the structure for the term phrase given in Figure 6. It is an example taken from their article, and exhibits stacking of relative clauses (the structure is simplified by omitting Comp's).

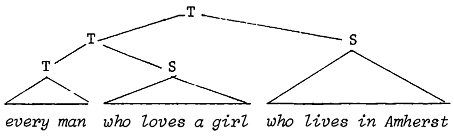


Figure 6

The translation of *every man* has to contain a variable for the relative clause. Recall that, in the conception of Bach & Cooper, the proposal discussed in Section 4.1 deals with the situation that we have the translation not containing *R*. Let us assume that we have taken the translation (21), which contains an unbound variable *R*.

(21) $\lambda P\forall x[man(x) \wedge {}^{\vee}R(x) \rightarrow {}^{\vee}P(x)]$.

Suppose now that the referent of *a girl* is to be contextually determined (this possibility is not considered by Bach & Cooper). Then the translation of a girl has to contain the variable R. Besides this variable the translation of (22) has to contain a variable R for the second relative clause. So the translation of (22) has to be (23).

(22) *who loves a girl*

(23) $\lambda z\exists y[girl(y) \wedge {}^{\vee}R(y) \wedge love_{*}({}^{\vee}z, {}^{\vee}y) \wedge {}^{\vee}R(z)]$.

Consequently, the translation of (24) has to be (25).

(24) *every man who loves a girl*

(25) $\lambda P\forall x[man(x) \wedge \exists y[girl(y) \wedge {}^{\vee}R(y) \wedge love_{*}({}^{\vee}x, {}^{\vee}y) \wedge {}^{\vee}R(x)] \rightarrow {}^{\vee}P(x)]$.

The translation of *who lives in Amherst* roughly is indicated in (26).

(26) $\lambda z[live\text{-}in\text{-}Amherst(z)]$.

The translation of the entire term-phrase in figure 6 is described by

(27) $\lambda R[every\ man\ who\ loves\ a\ girl']\ (who\ lives\ in\ Amherst')$.

This yields a logical expression which says that both the man and the girl live in Amherst, which is not the intended reading of the construction with stacked relative clauses.

These incorrect predictions are not restricted to stacking. The same problems arise in case a relative clause like *who runs* is combined with a disjoined term phrase like *the man or the woman*. Then semantically both terms are restricted, whereas syntactically only the second one is. The source of all these problems is that a single variable is used for relative clauses and for contextual restrictions. These two functions should, in my opinion, be separated. But then the left-over/not-there problem for relative clause variables arises with full force again.

2. As a motivation for interpreting the R's as contextual restrictions, the argument was given that when we speak about every man, we in fact intend every man from a contextually determined set. But this argument applies with the same force in case we speak about every man who runs. It is not true that terms sometimes are contextually determined, and sometimes not. If one wishes to formalize contextual influence, then every term should be restricted. This suggests (as under 1) a system of variables for context

restrictions which is independent of the system of variables for relative clauses.

3. Variables of which the interpretation is derived from the context have to receive a very special treatment. This can be shown most clearly by considering a sentence which has as translation a formula containing an occurrence of an unbound variable of the type of individuals or individual concepts: *he runs*, obtained from the sentence *he*$_n$ *runs*. These sentences have as translation $run(x_n)$. For every variable assignment this translation gets an interpretation. One of the possible assignments is that x_n is the person spoken to, so *He runs* would have the same truth conditions as *You run*. Some female person might be assigned to x_n, so the sentence may have the same truth-conditions as *she runs*. These are incorrect results, so there has to be some restriction on the variable assignments for x_n. There are also semantic arguments for such a restriction. A pronoun *he* usually refers to individuals from a rather small group (e.g. the person mentioned in the last sentence, or the person pointed at by the speaker). So again some restriction has to be given. These two sources of inadequacy can be dealt with by not evaluating a complete sentence with respect to all variable assignments, but only to a subset thereof. In the light of the arguments given above, this subset is rather small. So the contextually determined variables are not so variable at all; they behave more like constants.

4. A rather fundamental argument against the use of variables for formalizing contextual influence is the following. In PTQ the contextual factor of the reference point under consideration (a time world pair), is formalized by means of the so called indices I and J. Several authors have proposed to incorporate other factors in the indices. LEWIS (1970), for instance, mentions as possible indices: speaker, audience, segment of surrounding discourse, and things capable of being pointed at. These indices constitute an obvious way to formalize contextual influence. In the light of this, it is very important to realize that in IL the interpretation of constants is 'index dependent', whereas variables have an 'index independent' interpretation:

$$c^{A,i,j,g} = F(c)(i,j), \qquad x^{A,i,j,g} = g(x).$$

This means that in IL it is very strange to use logical variables for the purpose of encoding contextual restrictions. The obvious method is by means of constants. This is precisely the method employed in MONTAGUE (1968) and BENNETT (1978).

4.4. Conclusion

We considered Cooper's proposals concerning the solution of the 'not-there/left-over' problems. His idea to give a semantic treatment of the 'not-there' problem was not successfully formalized. His treatment of the variables 'left-over' led to incorrect results for English sentences. We have to conclude that the technical details of the Bach & Cooper proposal are such that their approach does not work correctly. This means that at the present stage of our investigations concerning the thematic question we are back at the situation of the end of Section 2: only the CN-S analysis seems to be possible.

I have not formally proved that it is impossible to find some treatment in accordance with Cooper's aims. As I said in Section 2, such a proof is, in general, difficult to give. But I have not only showed that the proposals by Bach & Cooper do not work correctly, I have also argued that they have to be considered as unsound. They constitute a very unnatural approach, and in my opinion one should not try to correct the proposals, but rather give up the idea underlying them altogether. Since I consider such proposals as unsound, I will in the next section put forward a principle which prohibits proposals of these kinds.

5. THE VARIABLE PRINCIPLE

In the previous section we have considered some attempts to deal with the 'not-there/left-over' problems. These attempts do not give me the impression that the considered situations they deal with are welcome; rather they seem to be escapes from situations one would prefer not to encounter at all. In my opinion these attempts arise from a neglect of the special character of syntactic variables. Syntactic variables differ from other words in the lexicon since they are introduced for a special purpose: viz. to deal with coreferentiality and scope. In this respect they are like logical variables, and in fact they can be considered as their syntactic counterpart. One would like to encounter syntactic variables only if they are used for such purposes. This special character of syntactic variables is expressed by the variable principle, of which a first tentative version is given in (29).

(29) *Syntactic variables correspond closely to logical variables.*

The intuition behind this statement is not completely new. THOMASON (1976) draws attention to the analogy between 'that-complement' constructions in English, and the λ-abstraction operator in logic. PARTEE (1979b) proposes the constraint that any syntactic variable must be translated into an expression of the logic containing an unbound logical variable. Partee does not accept this constraint the other way around, precisely because she does not want to disallow Cooper's treatment of free variables.

The formulation of the principle given in (29) is vague, and one might be tempted to strengthen it to (30).

(30) *An expression contains a syntactic variable if and only if its unreduced translation contains a corresponding unbound logical variable.*

This is intuitively an attractive formulation. However, a major drawback is that it does not fit into the framework of Montague grammar. It would give the unreduced translation of an expression a special status which it does not have in the framework as it is. The unreduced translation, would no longer be just one representation among others, all freely interchangeable. It would become an essential stage since the principle would have to function as a filter on it. It would no longer be allowed to reduce the intermediate steps in the translation process since then a semantically irrelevant occurrence of a logical variable might disappear, and thereby a translation that had to be rejected, might become acceptable. Therefore, I will give a formulation which turns the principle into a restriction on possible Montague grammars. The formulation below has the same consequences for the unreduced translation as (30), but it is not a filter on the unreduced translations and it leaves the framework untouched. This formulation is slightly more restrictive than (30), and than the formulation in JANSSEN (1980b).

The VARIABLE PRINCIPLE is defined as consisting of the following 6 requirements:

1a) *A syntactic variable translates into an expression which contains a free occurrence of a logical variable, and which does not contain occurrences of constants.*

1b) *This is the only way to introduce a free occurrence of a logical variable.*

2a) *If a syntactic rule removes all occurrences of a certain syntactic variable in one of its arguments, then the corresponding translation rule binds all occurrences of the corresponding logical variable in the translation of that argument.*

2b) *If a translation rule places one of its arguments within the scope of a binder for a certain variable, then its corresponding syntactic rule removes all the occurrences of the corresponding syntactic variable from the syntactic counterpart of that argument.*

3a) *The production of a sentence is only considered as completed if each syntactic variable has been removed by some syntactic rule.*

3b) *If a syntactic rule is used which contains instructions which have the effect of removing all occurrences of a certain variable from one of its arguments, then there indeed have to be such occurrences.*

This formulation of the variable principle is not what I would like to call 'simple and elegant'. I hope that such a formulation will be possible when the algebraic theory of the organization of the syntax is further developed. Suppose that we have found which basic operations on strings are required in the syntax (following PARTEE (1979a,b,see chapter 8)), and that a syntactic rule can be described as a polynomial over these basic operations. Then we may hope to formulate the variable principle as a restriction on the relation between the syntactic and semantic polynomials. We might then require that these polynomials are isomorphic with respect to operations removing/binding variables.

Requirement 1a) is a restriction on the translation of lexical elements. It can easily be checked whether a given grammar satisfies the requirement. It is met by all proposals in the field of Montague grammar that I know of; e.g. the PTQ translation of he_n is $\lambda P[{}^{\vee}P(x_n)]$, and the translation of the common noun variable one_n (HAUSSER 1979c) is the variable P_n.

For reasons of elegance, one might like to have formulation 1a') instead of formulation 1a).

1a') *A syntactic variable translates into a logical variable.*

In order to meet 1a') in the PTQ fragment, one could introduce a category of Proper Names containing *John, Mary*, he_1, he_2,... (with translations *john, mary*, x_1, x_2, respectively). Out of these Proper Names, Terms could be produced which obtain the standard translation ($\lambda P[{}^{\vee}P(john)]$, etc.). Since I do not know of a phenomenon, the treatment of which would be simplified using this approach, and since the variable principle then still would not have a simple formulation anyhow, I will not use it here. Requirement 1a) has as a consequence that the translation of a syntactic variable is logically *equivalent* to a logical variable. If constants are allowed to occur, then this would no longer be true (e.g. it is not true that for every c the formula $\exists x[x=c]$ is valid).

Requirement 1b) is a restriction both on the translation of lexical elements, and on the translation rules. This requirement is met by PTQ. It is not met by the proposals of BACH & COOPER (1978) which allow free variables to occur which do not have a syntactic counterpart. Since they do not present explicit rules, I do now know at which stage the context variable R is introduced, as a lexical ambiguity of the noun, or by means of some syntactic rule.

Requirements 2a) and 2b) are conditions on the possible combinations of a syntactic rule with a translation rule. Whether a grammar actually meets them is easily checked by inspection (PTQ does). Requirement 2b) is not met by the Bach & Cooper proposal since their approach in some cases gives rise to the introduction and binding of logical variables without any visible syntactic effect.

Requirements 3a) and 3b) we have already mentioned in chapter 6. They are not met by PTQ, nor by Bach & Cooper. In a certain sense they constitute the kernel of the principle. They express that certain configurations (described with respect to occurrences of variables) should not arise. When these requirements are met, the fundamental problems described in Section 1 disappear. As such, the two requirements are closely related to two instructions in JANSSEN (1980a, p.366), and to two conventions in RODMAN (1976, one mentioned there on p.176, and one implicitly used on p.170). Requirements 3a) and 3b) alone, i.e. without 1) and 2), would suffice to eliminate the syntactic side of the two fundamental problems, but then the close relationship between syntactic and logical variables would not be enforced. That freedom would give us the possibility to abuse syntactic variables for other purposes than coreferentiality and scope. An extreme case is given in JANSSEN (1980b), where some rules which obey 3a) and 3b), but violate 1) and 2), are defined in such a way that the information that a rule is obligatory is encoded in the syntactic variables. I intend to prohibit this and other kinds of abuse of variables by combining the third requirement with the first and second. In chapter 6, section 5.3, it is discussed how we might incorporate requirements 3a) and 3b) by filters and partial rules or by total rules (using a refined system of categories) For the present discussion it is irrelevant how these requirements are exactly incorporated in the system. Since we are primarily interested in the effects of the principle, it suffices to know that it can be done in some way.

Let me emphasize that the principle is intended to apply to the standard variables of intensional logic and their corresponding syntactic variables. For instance, the argument concerning the use of unbound variables for contextual influence does not apply if we do not translate into IL but into Ty2. If Ty2 is used, the variable principle does not simply apply to all the variables of type s. Neither does the principle apply to so called 'context variables' of HAUSSER (1979c), or the 'context expressions of GROENENDIJK & STOKHOF (1979), which both are added to IL for the special purpose of dealing with contextual influence.

The principle eliminates the basic problems from section 2 and disallows the treatment of variables aimed at in COOPER (1975), and COOPER (1979a,b). Another example of a treatment which is disallowed is the proposal of OH (1977). For a sentence without discourse or deictic pronouns he gives a translation containing a unbound variable! A consequence of the principle is that the denotation of a sentence is determined completely by the choice of the model and the index with respect to which we determine its denotation. In other words, the denotation is completely determined by the choice of the set of basic entities, the meaning postulates, the index, and the interpretation function for constants (i.e. the interpretations of the lexical elements in the sentence). In determining the denotation the non-linguistic aspect of an assignment to logical variables plays no role. This I consider to be an attractive aspect of the principle. What the impact of the principle is for the answer on the thematic question will be investigated in the next section.

6. MANY ANALYSES

6.1. The CN-S analysis for English

Do the rules for the CN-S analysis of relative clauses obey the variable principle?

Recall the PTQ rules from Section 2.1.

$S_{3,n}$ $CN \times S \to CN$

$F_{3,n}$ Replace he_n in β by *he/she/it* and him_n by *him/her/it*, according to the gender of the first CN in α; concatenate (α, *such that*, β).

$T_{3,n}$ (PTQ) $\lambda x_n[\alpha'(x_n) \wedge \beta']$.

This combination of $S_{3,n}$ and $T_{3,n}$ does not obey the variable principle since possible occurrences of x_n in α' are, by λx_n, bound in the translation, whereas the occurrences of the corresponding syntactic variable he_n in α are *not* removed. This aspect is the source of the 'collision of variables' mentioned in Section 3.1. (for details see section 3.4 of chapter 5). A reformulation of $T_{3,n}$ which avoids such a collision is given by THOMASON (1974, p.261).

$T_{3,n}$ (THOMASON)

$\lambda x_m[\alpha'(x_m) \wedge \tilde{\beta}']$

where $\tilde{\beta}'$ is the result of replacing all occurrences of x_n in β' by occurrences of x_m, where m is the least even number such that x_m has no occurrences in either α' or β'.

The syntactic rule $S_{3,n}$ removes the occurrences of he_n in β. Thomason's reformulation has the effect that the unbound logical variables x_n in β' do not occur free in the translation of the whole construction, whereas the same variables in α remain unbound. Nevertheless, Thomason's reformulation does not obey the variable principle since in the syntax occurrences of he_n in β are removed, whereas in the translation the occurrences of the corresponding variable (i.e.x_n) are not bound, but of a variable x_m (where $n \neq m$).

Another kind of objection against Thomason's rule is that it is not a polynomial over IL. This objection was considered in chapter 5, section 3.4. The formulation proposed there for the translation rule is the following.

$T_{3,n}$ $\lambda P[\lambda x_n[{}^{\vee}P(x_n) \wedge \beta'](^{\wedge}\alpha')]$.

This formulation has as a consequence that only those occurrences of x_n are bound, of which the syntactic counterparts are removed in $S_{3,n}$.

6.2. The S-S analysis for Hittite

Is an analysis of Hittite relative clause constructions possible which on the one hand satisfies the variable principle, and on the other hand produces such a construction out of two sentences?

Below I will describe an analysis which shows that the answer is affirmative. I will only deal with the example discussed in Section 3, and not with all other cases of Hittite relative clauses which are treated by COOPER (1975). My analysis is intended mainly as an illustration of the kinds of technique which are available if one obeys the variable principle.

The treatment described in Section 3 violates the variable principle because both subsentences in Figure 4 have a translation which contains an unbound variable, whereas the sentences themselves do not contain a syntactic variable. Given the principle, in both sentences there has to be an occurrence of a syntactic variable as well. The English variant of sentence S_2 gives a hint on how to do this. It contains in a CN-position the word (*one*) - probably added for explanatory reasons. This word suggests the introduction in the syntax of CN variables $one_1, one_2, \ldots$, which are translated into logical variables $P_1, P_2, \ldots$, respectively (such CN-variables are discussed in HAUSSER (1979c)). The rule which combines S_1 with S_2 will then give rise to a translation in which (by λ-conversion) the relevant property is substituted for P_n. In case one prefers not to introduce a new constituent one_n, a new variable of category T might be introduced alternatively: (31), translating as (32).

(31) $each_n$

(32) $\lambda Q[\forall x[{}^{\vee}P_n(x) \rightarrow {}^{\vee}Q(x)]$.

The variable in the translation of the relative clause can be introduced by the translation of the determiner *wh*. Therefore, the category of determiners (which contains the Hittite version of *every*, etc.) is extended with a variable (33), translating as (34).

(33) wh_n

(34) $\lambda Q \lambda P[{}^{\vee}Q(z_n) \wedge {}^{\vee}P(z_n)]$.

We have to combine a relative clause containing a free variable z_n with a main sentence containing a free variable P_n. This can be done by means of a single rule binding both logical variables and performing the relevant operations on both syntactic variables, or by means of two rules, each dealing with one variable at a time. The former method would yield the tree from figure 4, but it would implicate that a new kind of rules is introduced (rules with two indices). I will follow the two-rules approach.

First the relative clause is transformed into an expression of the new category Prop (=t//e), being a set of expressions denoting properties. We do this by means of the following rule (the numbers in the 800-series are numbers of newly proposed rules).

$S_{801,n}$ S → Prop

$F_{801,n}$ Replace wh_n in α by *wh*

$T_{801,n}$ $\lambda z_n[\alpha']$.

The rule combining a property with a sentence is

$S_{802,n}$ Prop × S → S

$F_{802,n}$ delete all occurrences of one_n from β; concatenate (α,β)

$T_{802,n}$ $[\lambda P_n \beta'](^\wedge\alpha')$.

Using these rules, the Bach & Cooper example is obtained in the way indicated in figure 7. Its translation is equivalent to the one given in Section 3 for figure 4. Since we assume that it is guaranteed that the variable principle is obeyed, no problems arise with the syntactic variables. The principle guarantees that rule $S_{802,1}$ is applied only in case the main sentence contains an occurrence of one_1 and that rule $S_{801,2}$ is applied only when the sentence contains an occurrence of the variable wh_2. Furthermore, it guarantees that all syntactic variables finally will have disappeared.

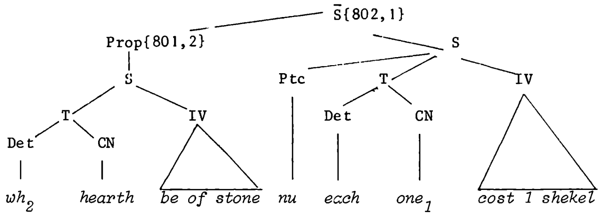


Figure 7

6.3. The T-S analysis for English

As shown in Section 6.2, an S-S analysis can be obtained simply by introducing a variable in the syntax, when such a variable is required in the translation. The same idea can be used to obtain a T-S analysis for relative clauses. In this case, we need a variable of the category Prop, written as *of kind*$_n$. It translates into the variable K_n.

A property and a common noun phrase combine to a new common noun phrase as follows:

S_{803} CN × Prop → CN

F_{803} concatenate (α,β)

T_{803} $\lambda y[\alpha'(y) \wedge \beta'(y)]$.

A category RC of relative clauses (RC = t///e) is introduced because RC's and Prop's will occur in different positions. The expressions of the category RC are made out of sentences as follows:

$S_{804,n}$ S → RC

$F_{804,n}$ delete the index n from all pronouns in α;
concatenate (*such that*, α)

$T_{804,n}$ $\lambda x_n[\alpha']$.

A relative clause may be quantified into a term phrase by substituting the relative clause for a property variable:

$S_{805,n}$ T × RC → T

$F_{805,n}$ substitute β for *of-kind*$_n$ in α

$T_{805,n}$ $\lambda K_n[\alpha'](^\wedge\beta')$.

An example of a production using these rules is given in figure 8.

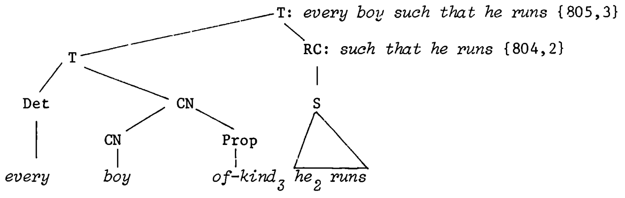

T: *every boy such that he runs* {805,3}
RC: *such that he runs* {804,2}
T
Det
CN
S
CN
Prop
every
boy
of-kind$_3$
he$_2$ *runs*

Figure 8

The translation of the lower term phrase in figure 8 is (35), the translation of the RC phrase (36), and of the upper term phrase (after reduction) is (37).

(35) $\lambda Q \forall x[boy(x) \wedge {}^{\vee}K_3(x) \rightarrow {}^{\vee}Q(x)]$

(36) $\lambda x_2[run(x_2)]$

(37) $\lambda Q \forall x[boy(x) \wedge run(x) \rightarrow {}^{\vee}Q(x)]$.

Note that the intermediate stage of an RC is not required if S_{805} is a double indexed rule, dealing both with he_n and $of\text{-}kind_m$.

6.4. The Det-S analysis for English

Is a Det-S analysis possible which obeys the variable principle? Recalling the pattern underlying the S-S and T-S analyses, one might try to find such an analysis as a variant of the CN-S analysis by introducing new variables. It appeared, to my surprise, that it is possible to obtain a Det-S analysis which is not a variant of the CN-S analysis, but which is a pure Det-S analysis (recall the proviso by Partee for her argumentation concerning the Det-S analysis). I will not discuss the heuristics of this analysis, but present the rules immediately.

$S_{806,n}$ Det × S → Det

$F_{806,n}$ remove all indices n from pronouns in β;
concatenate (α, *such that*, β)

$T_{806,n}$ $\lambda R[\alpha'({}^{\wedge}\lambda y[{}^{\vee}R(y) \wedge \lambda x_n[\beta'](y)])]$.

Maybe the following explanation of the translation is useful. A determiner δ is, semantically, a function which takes as argument the property η expressed by a noun and delivers a collection of properties which have a certain relation with η. S_{806} produces a determiner which takes a noun property η and delivers a set of properties which has that relation with the conjunction of η and the property expressed by the relative clause.

The combination of a CN with a Det-phrase, requires that the CN is placed at a suitable position in the determiner phrase. In the present fragment this position is the second position (if we had determiners like *all the*, then also other positions might under certain circumstances be suitable). The rule for this reads as follows:

S_{807} Det × CN → CN

F_{807} insert β after the first word of α

T_{807} $\alpha'({}^{\wedge}\beta')$.

The combination of the determiner *every* with the sentence *he*$_2$ *runs* yields determiner (38), with (39) as unreduced, and (40) as reduced translation.

(38) *every such that he*$_2$ *runs*

(39) $\lambda R[\lambda Q\lambda P[\forall x[{}^{\vee}Q(x) \rightarrow {}^{\vee}P(x)]](\lambda y[{}^{\vee}R(y) \wedge \lambda x_2[run(x_2)](y)])]$

(40) $\lambda R\lambda P\forall x[{}^{\vee}R(x) \wedge run(x) \rightarrow {}^{\vee}P(x)]$.

The combination of (38) the common noun *man* yields the term phrase (41), which has the (usual) reduced translation (42).

(41) *every man such that he runs*

(42) $\lambda P\forall x[man(x) \wedge run(x) \rightarrow {}^{\vee}P(x)]$.

The techniques which are used to obtain a T-S analysis from a CN-S analysis can be used as well to obtain a T-S analysis which is a variant of the Det-S analysis: introduce in the Det-S analysis the variable *of-kind*$_n$, but now within the determiner. This means that at least two kinds of T-S analyses are available.

6.5. Conclusion

In Section 5 a new principle was introduced: the variable principle. Obeying this principle we designed rules for relative clause constructions. It turned out that for English besides the CN-S analysis both the T-S and the Det-S analysis are possible in at least two essentially different variants. And for Hittite an S-S analysis is possible. So at the present stage of our investigations a negative answer to the thematic question has to be given: several analyses of relative clauses are possible.

Consider the T-S analysis of 6.3 again. Is it the kind of T-S analysis meant by Partee? I do not think so. At a certain level we indeed have a T-S analysis, but on another level in the production tree there is a CN-Prop analysis which is nothing but a variant of the CN-S analysis. The opposition between the two analyses was, however, the main point in the discussion of PARTEE (1973). So one could say that her conclusion that the pure T-S analysis cannot be used, in a certain sense still holds. For the case of Hittite however, the discussion primarily aimed at obtaining an S-S analysis at some level, rather than at avoiding the CN-S analysis on all levels. In Section 2 I quoted Bach & Cooper who expressed the hope for the 'happy discovery of yet unknown principles' which exclude the

T-S-analysis, but allow for the S-S-analysis. It seems reasonable to interpret this as the desire for a principle which prohibits the pure T-S analysis, but allows some variant of the S-S analysis. The variable principle has such an effect. But if it is interpreted as the hope for a principle which excludes all kinds of T-S analyses, or which allows a pure S-S analysis, then the variable principle is not such a principle. So the answer to the thematic question I gave above, has to be relativized: although *several* analyses are available, not *all* analyses are possible.

The answer to the thematic question obtained in this section, was based upon an investigation of the relative clause construction as such. Interaction with other phenomena was not taken into consideration. In the next section I will leave this isolation and consider the interaction of relative clause formation with some other phenomena.

7. OTHER ARGUMENTS

7.1. Syntax: gender agreement

The relative pronoun has to agree in gender with the antecedent nounphrase. In the Det-S analysis, this poses a problem. The rule which combines a determiner with a relative clause has to specify what is to be done with the syntactic variable. The formulation I gave of rule $S_{806,n}$ just deletes the index, so it gives a correct result if the noun has male gender. But in the same way as we produced *every boy such that he runs*, we may produce *every girl such that he runs*. It is not possible to formulate S_{806} in such a way that this kind of ill-formedness is avoided, because the information which gender the noun has, is not available at the stage at which the determiner and the relative clause are combined. Not removing the index would, according to the variable principle, require a free variable in the translation of the term phrase; but I do not see how this approach might work.

The T-S analysis gives rise to a similar problem. The rule which makes the relative clause (RC) out of a sentence (S), has to specify what has to be done with he_n. The formulation I gave of S_{804} works correctly for masculine nouns only. Again, information about the gender of the noun is not yet available, and not removing the index would constitute a break with the principle. This argument does not apply to the T-S analysis in which a double indexed rule is used. In the CN-S analysis, no problems arise from gender agreement, since at the stage at which the index has to be removed, the gender of the noun is known.

One should not conclude from this discussion that it is impossible to obtain correct gender agreement in case of the Det-S or T-S analysis under discussion. I expect that it can be done by means of further subcategorization. One has to distinguish feminine, masculine, and neuter relative clauses, and feminine, masculine, and neuter determiners, and probably one needs to make similar distinctions in other categories. Then the subcategory system provides the information needed to obtain precisely the correct combinations of relative clause, determiner and noun.

There is the hidden assumption in this discussion that gender agreement has to be handled within the syntax. If we do not assume this, then a phrase as *a girl such that he runs,* is no longer considered to be syntactically ill-formed. COOPER (1975) argues in favor of dealing with gender in the semantics (at least for English). Others might prefer to handle gender in pragmatics (Karttunen, according to PARTEE (1979a)). Then the arguments given here are no longer relevant. But in languages with grammatical gender (e.g. Dutch, German), this escape is not available. Here one might adopt one of the solutions I mentioned: refined subcategorization, a T-S analysis with a double indexed rule, or simply the CN-S analysis for relative clauses.

7.2. Semantics: scope

Consider the following sentence (exhibiting stacking on the head *man*):

(43) *Every man such that he loves a girl such that he kisses her is happy.*

This sentence has a possible reading in which *every* has wider scope than *a*. In a PTQ like approach (so with the CN-S construction for relative clauses), this reading is obtained by quantification of *a girl* into the CN phrase

(44) *man such that he loves* him_n *such that he kisses* him_n.

The corresponding translation of the sentence (44) reduces to

(45) $\forall y[\exists x[girl(x) \wedge man(y) \wedge love_*(^{\vee}y, ^{\vee}x) \wedge kiss_*(^{\vee}y, ^{\vee}x)] \rightarrow happy(y)]$.

Can this reading be obtained in other analyses of relative clauses?

In the T-S analysis this stacking of relative clauses can be obtained by means of a process indicated in figure 9. In order to obtain coreferentiality between both occurrences of the term him_n, the term *a girl* has to be substituted at a stage in which both relative clauses are present. The earliest moment at which this is the case, is immediately after the uppermost term has been formed. Using a rule analogous to the standard quantification rules would assign the existential quantifier wider scope than the universal quantifier, thus not yielding the desired reading. So it seems to be impossible to obtain in such a T-S analysis coreferentiality and correct scope at the same time.

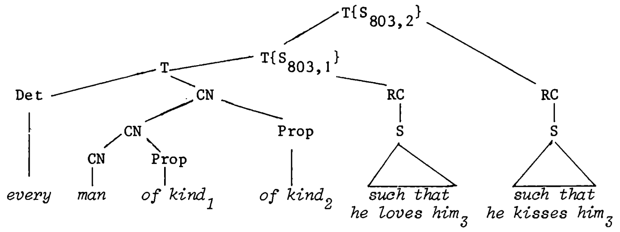


Figure 9

In the Det-S analysis the earliest stage at which the coreferentiality of *she* and *a girl* can be accounted for, is when the determiner phrase (46) has been formed.

(46) *every such that he loves* him_3 *such that he kisses* him_3.

Some later stage (e.g. the term level), might be selected as well. But in all these options, the quantification rule would give wider scope to *a* than to *every*, thus not yielding the desired reading.

Underlying this discussion is the assumption that there is something like stacking of relative clauses. If there is stacking, then the rule for quantification into a CN is essential for the PTQ fragment (FRIEDMAN & WARREN (1979b)). But is stacking indeed a phenomenon of natural language? As for Hittite, BACH & COOPER (1975) inform us that no stacking occurs, and in Dutch and German stacking is not possible. As for English, no author expresses doubts, except for PARTEE (1979b). She states that the evidence for stacking is spurious. If we accept this, it would leave a rather small basis for our argumentation concerning an answer on the thematic question.

There is another phenomenon, however, that requires quantification into CN's. It might be the kind of examples meant by PARTEE (1975, p.236). Example (47) assumes that there are common nouns in the fragment of the form *friend of*.

(47) *Every picture of a woman which is owned by a man who loves her is a valuable object.*

Here the intended reading is the one in which *every* has wider scope than *a*, and in which there is coreferentiality between *a woman* and *her*. This reading can easily be obtained by means of substitution of *a woman* into the CN-phrase (48).

(48) *picture of* he_1 *such that it is owned by a man such that he loves* him_1.

So even if we do not accept stacking as a phenomenon of English, a CN-S analysis would be required.

It is remarkable to notice that the variable principle plays no role in the discussion concerning scope. The occurrences of the Prop variables, which form a practical consequence of the principle, were not relevant. If they were omitted, which would bring us back to the original Bach & Cooper approach, then still the same problems would arise with respect to scope. So even without the variable principle a CN-S analysis appears to be required. This conclusion has to be relativized immediately. I have not given a formal proof that it is impossible to obtain a correct treatment of scope in the other analyses. I just showed that the CN-S analysis provides a direct basis for a semantic treatment of scope phenomena in a way that the considered T-S and Det-S analyses can not. This conclusion mentions another argument for relativizing. We only considered the three analyses which had our main interest. A lot more analyses are possible, and for some a correct treatment of scope may be possible. For instance, a correct treatment of scope might be possible if the category of determiners contains variables for which a determiner can be substituted in a later stage.

7.3. Conclusion

In the previous section we observed that the framework of Montague grammar hardly restricts the possible syntactic analyses of relative clauses. In this section we investigated the possibilities for incorporating the available options in a somewhat larger fragment. It turned out that from the three main options only one was suitable. From this we learn that it is important to consider phenomena not only in isolation, but to design grammars for larger fragments. The fact that for each isolated phenomenon there are many syntactic options available, gives us a firm basis for the hope that it is indeed possible to find a combination of syntactic constructions that fits together in a system yielding the correct semantics for the constructions involved. Thus we see that extending fragments is a fruitful step which has impact on the description of isolated phenomena. This can be considered as a reaction to be a remark Van BENTHEM (1981, p.31) who denies the use of generalization and the combination of partial theories.

8. THE GENERAL QUESTION

In this section I answer the general version of the thematic question. We employ a framework in which the syntax and semantics have to be algebras, and in which meaning assignment is a homomorphism. The general version of the thematic question was to what extent this organization of the grammar restricts the options we have available for describing a particular phenomenon in the syntax.

For the special case of relative clause formation we obtained in section 6 the answer that any kind of analysis can be obtained, but that certain kinds of analysis cannot be avoided. This practical result will be explained below on the basis of the algebraic properties of the framework, and the result will be generalized to an answer on the general question.

Let us suppose that we have found a semantic operation T_{888} which takes two arguments, and delivers the meaning of a certain construction. So in the semantics we have the construction step $T_{888}(\alpha',\beta')$. Due to the homomorphism relation, there has to be a corresponding operation $F_{888}(\alpha,\beta)$ in the syntax, and the two semantic arguments have to correspond with the two syntactic arguments. Instead of the semantic step $T_{888}(\alpha',\beta')$, several variants are possible, each with its own consequences for the syntax. These variants amount to a construction process with two stages. We may first have $T_{888}(\alpha',R)$, where R is a variable, and introduce in a later stage a λ-operator for R taking β' as argument:

$$\lambda R[\ldots T_{888}(\alpha',R)\ldots](\beta').$$

This means that the syntactic expression β can be introduced in an arbitrary later stage of the syntactic production process. Consequently, a lot of variants of the original syntactic construction can be formed. These variants are based on the use of the construction step T_{888} (α',R) in the logic. Due to the variable principle, the variable R has to be introduced by the translation of some syntactic variable. Let us suppose that V is such a variable. Due to the homomorphic relation between syntax and semantics, this means that in the syntax there has to be a step F_{888} (α,V). So whereas we have gained the freedom to introduce β in a later stage of the syntactic construction process, step F_{888} is not avoided. The same argumentation applies when the first argument of T_{888} is replaced by a variable. It is even possible to replace both arguments by a variable, thus obtaining a large freedom in the syntax concerning the stage at which α and β are introduced. But in all these variants F_{888} is not avoided. Application of this argumentation to the case of relative clauses (where two basic constructions are found) means that we cannot avoid both the CN-S and the Det-S construction at the same time.

So on the basis of the compositionality principle, formalized in an algebraic way, many relative clause constructions are possible. This is due to the power of λ-abstraction. This operation makes it possible that on the semantic side the effect is obtained of substituting the translation of one argument on a suitable position within the other argument, whereas in the syntax a completely different operation is performed. Referring to this power Partee once said 'Lambdas really changed my life' (Lecture for the Dutch Association for Logic, Amsterdam, 1980).

The above argumentation is not completely compelling: there is (at least) one exception to the claim that it is not possible to make a variant of a given semantic construction which avoids the corresponding syntactic construction step. An example of such an exception arose in the S-S analysis for Hittite. In the main sentence we had the Det-CN construction *each* one_n, where one_n was a variable. We obtained a variant in which there is no Det-CN construction: the logical variable introduced by one_n, could be introduced by a new variable $each_n$ (see (34)). The algebraic description of this method is as follows. Consider again T_{888} (α',R). The variable R might, under certain circumstances, be introduced by the translation of α, thus allowing to replace T_{888} by a related semantic operation which takes only one argument. That the translation of α introduced the variable R, means that in the syntax α is to be replaced by some variable, say an indexed variant of α. Its translation is then a compound expression (being a combination of the old translation α' with the variable R). This process, which avoids to have F_{888} in the syntax, is possible only if α is a single word with a translation which does not contain a constant (e.g. if α is a determiner). If the translation of α would contain a constant, then requirement 1a) of the variable principle would prohibit that its translation introduces a variable. If α is not a single word, then it cannot be replaced by a syntactic variable (maybe one of its parts can then be indexed). This method of creating exceptions would be prohibited when requirement 1a) of the variable principle would be replaced by the more restrictive version 1a'). In order to prove that the exception described here is the only one by which a given analysis can be avoided, the details of the relation between operations in the semantics or in the syntax have to be formalized algebraically (see also Section 3).

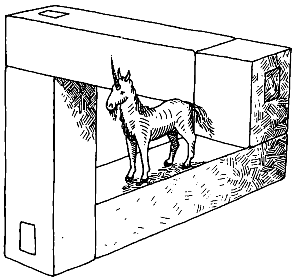
These algebraic considerations explain the results of our practical work. On the basis of these considerations it would be possible to explain that a Det-S analysis which is variant of the CN-S analysis, is not to be expected (in any case the described method for obtaining variants does not work). The algebraic considerations also answer the general question whether the principle of compositionality restricts the options available for descriptive work. On the basis of a given construction step, a lot of variants are possible, but due to the variable principle and the homomorphic relation between syntax and semantics, this construction step cannot be avoided in these variants. So the answer to the general question is that there are indeed restrictions on the syntactic possibilities, but only in the sense that a basic step cannot, generally speaking, be avoided. But these restrictions are not that strong that only a single analysis is possible. Formal proofs for these considerations would require, as I said before, a further algebraization of the syntax.

CHAPTER X

SCOPE AMBIGUITIES OF TENSE, ASPECT AND NEGATION

ABSTRACT

In this chapter verbal constructions with *will*, *have*, with negation, and with the past tense are considered. The meanings of these syntactic constructions are expressed by semantic operators. These operators have a certain scope, and differences in scope give rise to semantic ambiguities. These scope ambiguities are investigated, and a grammar dealing with these phenomena is presented. In this grammar features and queries are used, and the grammar produces labeled bracketings.



1. INTRODUCTION

Verbal constructions with *will, have,* with negation, or with the past tense, give rise to semantic operators: negation, tense operators and aspect operators. The syntactic counterparts of such operators I will call 'verb-modifiers'. So a basic verb modifier consists sometimes of a single word (*will, have*), sometimes of two words (*do not*), and sometimes of a verb affix (for the past). Compound verb modifiers are combinations of basic modifiers; they may consist of several words (*will not have*).

The semantic operators which correspond with basic verb modifiers have a certain scope, and a sentence can be ambiguous with respect to the scope of such an operator. The aim of the present chapter is to present a treatment of scope phenomena involving terms and verb modifiers. Examples of such ambiguities are provided by sentences (1) and (2). Both are ambiguous; each sentence may concern either the present president or the future president.

(1) *The president will talk*

(2) *John will hate the president.*

It is not my aim to analyse in detail the semantic interpretation of operators corresponding with verb modifiers. I will not present proposals for the formal semantics of tense or aspect; there is, in my treatment, no semantic difference between past and perfect (there is a syntactic difference). It is my aim to investigate only scope phenomena of operators and not to consider the operators themselves.

The main conclusion concerning the treatment of scope will be that the introduction of verb modifiers has to be possible both on the level of verb phrases and on the level of sentences. Another conclusion will be that compound verb modifiers have to be introduced in a step by step process: each step introducing one semantical operator. The treatment that I will present does not deal with all scope phenomena correctly (see section 5).

2. THE PTQ APPROACH

2.1. Introduction

As starting point for my investigations I take the treatment of verb modifiers as presented in MONTAGUE 1973 (henceforth PTQ). I will discuss syntactic and semantic aspects of that proposal and compare these with related aspects of my approach.

2.2. Syntax of PTQ

The grammar of PTQ has six operations for the treatment of verb modifiers: rules for present, perfect and future, and for the negated variants of these tenses. Some examples:

F_{14} (*John, walk*) = *John has walked*

F_{15} (*John, walk*) = *John has not walked*

F_{11} (*John, walk*) = *John does not walk.*

These operations are completely independent. The operation 'make a sentence in the perfect tense' is independent of the operation 'make a sentence in the negative perfect tense'. One would like to have here another situation. My treatment aims at a so-called 'orthogonal' syntax: each phenomenon will be treated by its own collection of rules (e.g. 'negating' will be treated by means of a rules which just deal with negation), and all such collections of rules will have the same structure as much as possible.

The PTQ rules do not treat conjoined verb phrases correctly since only the first verb is conjugated. So the PTQ syntax produces (3) instead of (4).

(3) *John has walked and talk*

(4) *John has walked and talked*

FRIEDMAN (1979) has given a treatment of this kind of error, and the treatment in this chapter of these problems will be about the same as hers.

The rules of PTQ deal with only three verb-modifiers: future, perfect and negation. Compound modifiers such as past perfect (in *had walked*) are not treated, nor the simple past (*walked*). These modifiers will be incorporated in the fragment of the present chapter. Furthermore, compound verb phrases will be incorporated of which the conjuncts (disjuncts) may be modified in different ways (*has walked and will talk*).

2.3. Ambiguities

The grammar of PTQ deals with several scope ambiguities. I will recall two of them because variants of them will return in the discussion. The most famous example is (5). This sentence has a de-re reading (6) and a de-dicto reading (7).

(5) *John seeks a unicorn*

(6) $\exists u[unicorn_*(u) \wedge seek_*(john,u)]$

(7) $seek({}^\wedge john, {}^\wedge\lambda P\exists u[unicorn_*(u) \wedge {}^\vee P(u)])$.

Another example is the scope ambiguity in (8); this sentence has readings (9) and (10)

(8) *Every man loves a woman*

(9) $\forall u[man_*(u) \rightarrow \exists v[woman_*(v) \wedge love_*(u,v)]]$

(10) $\exists v[woman_*(v) \wedge \forall u[man_*(u) \rightarrow love_*(u,v)]]$.

The readings of (8) have a remarkable property. Reading (10) logically implies (9). This means that there is no situation in which (10) is true, while (9) is false. For this reason one might doubt whether this scope ambiguity is an ambiguity we have to account for: reading (9) seems to be always acceptable. I will give two arguments explaining why (8) is considered ambiguous. Both arguments are due to Martin Stokhof (personal communication); see also chapter 4, section 6.

The first argument is that for slight variants of (8) the two readings are logically independent. Consider sentence (11), in which we understand *one* as *precisely one*.

(11) *Every man loves one woman.*

This sentence has readings (12) and (13), where neither (12) follows from (13), nor (13) from (12).

(12) $\forall u[man_*(u) \rightarrow \exists! v\ woman_*(v) \wedge love_*(u,v)]]$

(13) $\exists! v[woman_*(v) \wedge \forall u[man_*(u) \rightarrow love_*(u,v)]]$.

A more well-known variant of the scope ambiguities of (8) and (11) is sentence (14). Also here the two readings are independent.

(14) *Every man in this room speaks two languages.*

These considerations show that sentences closely resembling (8) exhibit independent ambiguities.

The second argument is that in certain contexts the weaker reading of (8) is required. Consider (15) or (16).

(15) *It is not the case that every man loves a woman*

(16) *John does not believe that every man loves a woman.*

Sentence (15) can be used in situations in which it means (17), as well as in situations where it means (18).

(17) $\neg[\forall u[man_{*}(u) \rightarrow \exists v[woman_{*}(v) \wedge love_{*}(u,v)]]]$

(18) $\neg\exists v[woman_{*}(v) \wedge \forall u[man_{*}(u) \rightarrow love_{*}(u,v)]]$.

Here the implication goes in the other direction: reading (17) implies (18). So if we prefered to have only one reading for (15), it would have to be (18). It is very likely that (15) is obtained by building (8), and next negating it. This construction requires that reading (18) of (15) be produced from reading (10) of (8). So sentences like (15) require that reading (10) is available. Hence (8) should get both reading (9) and (10).

2.4. Model

In several recent proposals arguments are put forward in favor of another model for time than the one used in PTQ. Such proposals are based upon a model with an interval semantics for time, rather than one with time point semantics (DOWTY 1979b, many contributions in ROHRER 1980). I will not incorporate these innovations, but follow the PTQ logic and semantics since it was not my aim to improve the PTQ interpretation of modifiers. This means that the logic does not provide for the tools to discriminate semantically between simple past and perfect, and therefore I will assign the same meanings to them. The use of the PTQ model has as a consequence that, formally spoken, I only deal with a limited use of tense: the reportive use (see BENNETT 1977).

Using such a 'primitive' semantics is, for the present purposes, not a great drawback. The scope phenomena under discussion will arise within any semantic treatment of tenses, no matter what kind of a model is used. I expect that my treatment can be adopted for another model (by taking the same derivational history, but changing the translations or their interpretations).

3. BASIC VERB MODIFIERS

In this section sentences will be considered which contain basic verb modifiers. First such sentences will be considered from a syntactic point of view. The PTQ rules produce such modifiers in few contexts only, but there are more situations in which they may occur. Next we will consider such sentences from a semantic point of view and investigate their scope ambiguities. Finally we will consider which kind of rules might be used to produce such sentences and to obtain the desired meanings.

The first kind of situation we will consider are the complements of verbs like *try*, *assert* and *regret*. The rules of PTQ allow for unmodified verb phrases as complement. An example is (19).

(19) *John tries to run.*

PTQ does not allow for negated verbs as complement. Such complements are possible as is pointed out by BENNETT (1976); see example (20). The sentence is intended in the reading that what John tries, is not to run.

(20) *John tries not to run.*

As sentence (21) shows, complements in the perfect are also possible (unlike the PTQ predictions).

(21) *John hopes to have finished.*

Future is not possible in these complements (but in Dutch it is possible).

The second kind of situations where the PTQ rules are inadequate is provided by sentences with conjoined verb phrases. The PTQ syntax states that the first verb has to be conjugated. If we assume that the rule is changed to mark all relevant verbs, then sentences like (22) are produced.

(22) *John has walked and talked.*

In the PTQ approach it is not possible to obtain differently modified verbs in the conjuncts to the verb phrase; yet it was noticed by BACH (1980) and JANSSEN (1980b) that they can be combined freely. Some examples, due to Bach (op. cit.) are (23) and (24).

(23) *Harry left at three and is here now.*

(24) *John lives in New York and has always lived there.*

These examples can easily be adapted for other verb modifiers. In (25) negation occurs and in (26) future.

(25) *Harry left at three but is not here now.*

(26) *John has always lived in New York and will always stay there.*

So the PTQ syntax has to be extended for complements and conjuncts.

Now we come to the semantic aspect. Sentences which contain a modifier exhibit scope ambiguities with respect to the corresponding operator. An example is (27).

(27) *The president will talk.*

This sentence has a reading which says that the present president will speak at a moment in the future (maybe after his presidency). It also has a reading which says that on a future moment the then president will speak. So sentence (27) has readings (28) and (29).

(28) $\exists u[\forall v[president_*(v) \leftrightarrow u = v] \wedge W[talk_*(u)]]$

(29) $W\exists u[\forall v[president_*(v) \leftrightarrow u = v] \wedge talk_*(u)]$.

Notice that I consider *president* a predicate which may apply for different reference points to different persons. In some cases an index independent interpretation of an in principle index-dependent noun seems to be required. The American hostages in Iran will probably always be called *hostages* although they are no longer hostages. This means that this noun in sentence (30) is used with an index independent interpretation.

(30) *The hostages were received by the president.*

I assume that even *the president* can be used in an index independent way; in a biography about Eisenhower one might say

(31) *The president studied at West-Point.*

With an index independent interpretation of *president* formulas (28) and (29) are equivalent. An example of a term for which only an index-dependent interpretation is possible is *70-years-old-man*. Sentence (32) only has readings (33) and (34).

(32) *A 70 years old man will visit China.*

(33) $W\exists u[70\text{-}years_*(u) \wedge man_*(u) \wedge visit_*(u,China)]$

(34) $\exists u[70\text{-}years_*(u) \wedge man_*(u) \wedge W[visit_*(u\ China)]]$.

For past tense and for negation ambiguities arise which are related to the ambiguities for future discussed above. For perfect the opinions vary. Some native speakers have claimed that perfect can only have narrow scope, whereas others have no problems with two readings for sentence (35).

(35) *The president has talked.*

The grammar I wil present, assigns two readings to (35), but a slight modification would give only one reading.

For sentences with differently modified verb phrases there is no scope ambiguity. Sentence (36) only has reading (37), see BACH 1980.

(36) *A woman has walked and will run*

(37) $\exists u[\mathit{woman}_*(u) \wedge \mathrm{H}[\mathit{walk}_*(u)] \wedge \mathrm{W}[\mathit{run}_*(u)]]$.

The above examples concerning embeddings and conjunctions suggest that it is useful to have rules which produce modified verb phrases. This is the approach that will be followed in this article. But the examples do not *prove* that it is impossible to design a system in which only sentences with verb modifiers are produced, and no modified verb phrases. I will sketch below some problematical aspects of such approaches.

One might think of introducing the perfect on the level of sentences, thus obtaining (39) from (38). Combination with (40) then yields (41).

(38) *Harry leaves at three*

(39) *Harry has left at three*

(40) *Harry is here now*

(41) *Harry has left at three and Harry is here now.*

From (41) we obtain (42) by means of a deletion rule.

(42) *Harry has left at three and is here now.*

For these sentences there arise no problems with this approach. But for (43) it is problematic since (43) does not have the same meaning as (44).

(43) *A man left at three and is here now*

(44) *A men left at three and a man is here now.*

Our framework requires that there be a semantic operation which corresponds with the rule that produces (42) from (41) and (43) from (44). I do not know of a semantic operator which has the desired effect, and therefore it is questionable whether this approach can be followed.

A variant of this method, due to Van Benthem (personal communication) is to produce (42) from (45).

(45) *He$_1$ has left at three and is here now.*

Sentence (45) is produced in the way sketched above, so obtained from (46) by means of a deletion rule.

(46) *He$_1$ has left at three and he$_1$ is here now.*

The semantic problem mentioned above does not arise because (45) and (46) are equivalent. I expect that an approach like this will require rules which are far more complex than the rules which produce modified verb phrases in this chapter.

If we have rules introducing verb modifiers at the level of verb phrases do we then still need rules introducing them at the level of sentences? The answer of BACH (1980) seems to be that only rules for verb phrases are needed. He presents a new translation rule corresponding with the syntactic rule which produces a sentence from a term and a verb phrase. His translation rule has the effect that in the translation of the sentence the operator in the IV-translation gets wider scope than the subject. So the basic situation is that subjects obtain narrow scope, and subjects can obtain wide scope by quantifying in. In this way the two readings of (47) are obtained.

(47) *The president will walk.*

An exception to this pattern is the conjunction (disjunction) of differently modified verb phrases. As we observed above, the subject can only have wide scope. Recall (36)

(36) *A woman has walked and will run.*

In order to deal with such constructions, Bach presents translation rules for conjunction and disjunction of verb phrases which have the effect that for such constructions the subject gets wide scope.

Bach's approach is insufficient because there are examples where one whishes to quantify a term in, but where nevertheless this term should be within the scope of the tense operator. I will give three examples. Each exhibits in the future tense a phenomenon for which quantification rules are commonly used in the present tense. The first example concerns scope ambiguity of quantifiers: sentence (48) with reading (49).

(48) *Every catholic will follow a man*

(49) $W\exists u[man_*(u) \wedge \forall v[catholic_*(v) \rightarrow follow_*(v,u)]]$.

In order to obtain reading (49) one wishes to quantify *a man* into *Every catholic follows him* and only after that, assign the tense. The second example concerns the de-dicto/de-re ambiguity: sentence (50) with reading (51).

(50) *John will seek a unicorn*

(51) $W\exists u[unicorn_{*}(u) \wedge seek_{*}(john,u)]$.

Here John seeks a future 'de-re unicorn'. Again one wishes to quantify in, and then assign tense. The third example concerns coreferentiality of terms inside the scope of the tense operator: sentence (52) with reading (53).

(52) *The president will love a woman who kisses him*

(53) $W\exists u[\forall v[president_{*}(v) \leftrightarrow u = v] \wedge \exists w[woman_{*}(w) \wedge kiss_{*}(w,u) \wedge love_{*}(u,w)]]$.

This translation represents the reading in which the loving and kissing happen on the same moment in the future. Again one wishes to produce this sentence by means of first quantifying in at the sentence level, followed by tense assignment on that level. Related examples can be given for other tenses and aspects.

For the introduction of negation on the sentence level related examples can be given: situations where one wished to quantify in, but where negation has wide scope. Examples are the wide quantifier scope in (54), the de-re reading of (55) and the coreferentiality in (56).

(54) *Every woman does not love a man*

(55) *John does not seek a unicorn*

(56) *The president does not love the woman who kisses him.*

The main conclusion of this section is that rules are needed which introduce modifiers on the level of verb phrases, but that also rules are needed which do so on the level of sentences. This aspect constitutes the fundamental difference between the present approach and the approach of BACH (1980). Notice that an important part of the argumentation is based upon the fact that phenomena like scope of terms, de-dicto/de-re ambiguity and coreferentiality, are dealt with by means of quantification rules.

The last part of this section consists of two examples of sentences which are produced according to the ideas I have just sketched. The details of the rules will not be given here, but the sequence of stages of the process (and the respective translations) are the same as the ones obtained by using the rules of the grammar from section 7.

The first example is sentence (57), with reading (58).

(57) *John will seek a unicorn*

(58) $W[\exists u[unicorn_*(u) \wedge seek_*(john,u)]$.

First sentence (59) is produced, it has (60) as translation.

(59) *John seeks him$_1$*

(60) $seek(^\wedge john, ^\wedge\lambda P ^\wedge P(x_1))$.

The next step is to quantify in the term *a unicorn*. Then sentence (61) is obtained, with translation (62).

(61) *John seeks a unicorn*

(62) $\exists u[unicorn_*(u) \wedge seek_*(john,u)]$.

The last step is the introduction of future tense in (61). This gives us sentence (57), with (58) as translation.

The second example concerns the sentence *John tries not to run*. This sentence contains the verb phrase *not to run*, and this raises the question which kind of translation we use for verb phrases. BACH (1980) has given several arguments for considering verb phrases as functions operating on subject terms. This approach has as a consequence that a new, somewhat complex translation rule has to be used for S4 (the rule which combines a T and an IV to make an S). One of Bach's arguments in favor of considering verb phrases as functions was his treatment of tense and aspect. As we concluded, his proposal is in this respect not satisfactory. His other arguments in favor of verb phrases as functions, concern phenomena I do not deal with in this article (such as 'Montague phonology' and constructions like *A unicorn seems to be approaching*). Since in our fragment we will not have any of the advantages of that approach, I will use the simpler PTQ translation. But no matter which translation is chosen, the conclusion that modifiers have to be introduced on two levels remains valid.

Let us return to the example, sentence (63) with translation (64).

(63) *John tries not to run*

(64) $try\ to(^\wedge john, \lambda x \neg[(run(^\vee x)])$.

The first stage in the production of this sentence is to produce verb phrase (65). Its translation as explained above, is (66).

(65) *do not run*

(66) $\lambda x \neg [run_*(^\vee x)]$.

The next step is the addition of *try to*, yielding (67), with (68) as translation.

(67) *try not to run*

(68) $try\ to(\lambda x, \neg [run_*({}^{\vee}x)])$.

Combination with the term *John* gives sentence (63), with translation (64).

4. COMPOUND VERB MODIFIERS

In this section I will consider sentences in which verbs occur which are accompanied by compound modifiers: constructions with *will not, will have, had, would,* etc. The sentences exhibit ambiguities which give us suggestions as to how to deal with compound modifiers.

The first example concerns the combination of negation and future. Sentence (69) has three readings, viz. (70), (71) and (72).

(69) *Every woman will not talk*

(70) $\forall u[woman_*(u) \rightarrow \neg W[talk_*(u)]]$

(71) $\neg W\forall u[woman_*(u) \rightarrow talk_*(u)]$

(72) $\neg \forall u[woman_*(u) \rightarrow W\ talk_*(u)]]$.

Notice that in all readings negation has wider scope than future. The first two readings are the most likely ones. A situation in which the relative scope of the third reading seems to be intended arises in HOPKINS (1972, p.789). Hopkins argues that it is not necessary to always design elegant computer programs because

(73) *Every program will not be published. Many will be used only once.*

In the PTQ approach only readings (70) and (71) can be obtained. This is due to the fact that in PTQ tense and negation are treated as an indivisible unit which is introduced by one single step.

For sentence (74) related ambiguities arise. The sentence is three ways ambiguous.

(74) *The president will have talked.*

This sentence may concern

(i) An action of the present president (maybe after his presidency).

(ii) An action of some president during his presidency (maybe a future president).

(iii) An action of a future president (maybe before his presidency).

This readings are presented in (75), (76) and (77) respectively.

(75) $\exists u[\forall v[president_*(v) \wedge u = v] \wedge \mathrm{WH}[talk_*(u)]]$

(76) $\mathrm{WH}\exists u[\forall v[president_*(v) \leftrightarrow u = v] \wedge talk_*(u)]$

(77) $\mathrm{W}\exists u[\forall v[president_*(v) \leftrightarrow u = v] \wedge \mathrm{H}[talk_*(u)]]$.

I assume that (75) is the most likely reading of (74). The relative scope of the tense operators and president as indicated in (76) is, however, the most likely reading of (78)

(78) {*In 2000 the political situation will be different since*}
A USA president will have visited Cuba.

The relative scope as indicated in (77) is the most likely reading of (79).

(79) *The president will have learned Montague grammar at high school.*

These examples show that the two tense operators corresponding with the compound modifier 'future perfect' may have different scope. For the other compound modifiers related examples can be given. I will give some examples of the reading in which the scope of the two operators is not the same. Sentence (80) with reading (81) can be said about Eisenhower.

(80) *The president had been a general.* {*Therefore he knew about the power of the military-industrial complex*}

(81) $\mathrm{H}\exists u[\forall v[president_*(v) \leftrightarrow u = v] \wedge \mathrm{H}[general_*(u)]]$.

Sentence (82) gives information about the former Dutch queen Wilhelmina.

(82) {*In May 1940 Wilhelmina had to leave her country but*}
The queen would return to Holland.

(83) $\mathrm{H}\exists u[\forall v[queen_*(v) \leftrightarrow u = v] \wedge \mathrm{W}[return_*(u)]]$.

Notice that in sentence (82) *would* is used to indicate a certain temporal sequence. At the moment in the past under consideration, the return was still in the future. Also the construction *would have* can be used to describe a certain temporal sequence. The information about queen Wilhelmina given above can be extended by (84).

(84) *At her departure she was just the queen, at the moment of her return she would have become a symbol of patriotism.*

The use of *would* and *would have* described above is somewhat exceptional. More frequently they are used in constructions like (85) and (86).

(85) *John would come, but he is not here.*

(86) *If John had come, we would have won the game.*

I do not intend to deal with constructions like (85) and (86), but only with the 'temporal' constructions.

For simple modifiers ambiguities of the kind considered above do *not* arise: (87) does *not* have reading (88), which would express the fact that the action may take place after the presidency of a future president.

(87) *The president will visit Holland*

(88) $W\exists u[\forall v[president_*(v) \leftrightarrow u = v] \wedge W[visit_*(u, Holland)]]$.

The ambiguities considered in this section suggest that the compound modifiers have, for semantic reasons, to be introduced in a process with several stages, each stage introducing one operator in the translation. For instance, a sentence containing *will have* is obtained by first introducing perfect and next introducing future. Analogously *had* is analyzed as past + perfect and *would* as past + future. The semantic ambiguities can easily be accounted for since for an operator we have the options of introducing it on the level of verb phrases and of introducing it on the level of sentences.

Besides the semantic arguments there is also syntactic evidence for the introduction of compound modifiers by means of a process with several stages. Some compound modifiers can be split up when they occur in connection with a conjoined verb phrase. An example is (89).

(89) *The president will have talked and have walked.*

In (89) the verb phrases *have talked* and *have walked* share the auxiliary verb *will*.

The main conclusion of this section is that compound modifiers have to be introduced by a process with several stages, each stage introducing a new operator in the translation. An example illustrating this process is sentence (90) with reading (91).

(90) *The president will have talked*

(91) $W\exists u[\forall v[president_*(v) \leftrightarrow u = v] \wedge H[talk_*(u)]]$.

The first step is the production of the verb phrase (92), which has translation (93).

(92) *have talked*

(93) $\lambda x \mathrm{H}[talk_{*}({}^{\vee}x)]$.

Next sentence (94) is formed with translation (95).

(94) *The president has talked.*

(95) $\exists u[\forall v\ president_{*}(v) \leftrightarrow u = v] \wedge \mathrm{H}[talk_{*}(u)]]$.

The last step is the introduction of the future, this yields sentence (90) with translation (91).

5. COMPLEX CONSTRUCTIONS

5.1. Introduction

In the previous sections we considered scope phenomena of simple and of compound verb modifiers. In this section scope phenomena will be considered in connection with more complex constructions than considered before. The most important ones are conjoined and disjoined phrases and combinations of them. I will use the name conjoined phrases to cover such conjunctions and disjunctions except where the difference is relevant. This section has a somewhat different character than the previous two, because conjoined constructions give rise to phenomena which do not constitute a clear and simple pattern. The acceptability of the sentences or interpretations is sometimes marginal and the judgements may have to be changed in some cases. The present discussion is intended primarily to point out some interesting phenomena.

5.2. Conjoined verb phrases with positive verbs

Conjoined verb phrases which consist of unmodified verbs give rise to the same phenomena as single verbs. The conjoined phrases can be modified as if they were simple verbs and they exhibit the same ambiguities. An example is sentence (96), which has readings (97) and (98).

(96) *The president will walk and talk*

(97) $\mathrm{W}\exists u[\forall v[president_{*}(v) \leftrightarrow u = v] \wedge [walk_{*}(u) \wedge talk_{*}(u)]]$.

(98) $\exists u[\forall v[president_{*}(v) \leftrightarrow u = v] \wedge \mathrm{W}[walk_{*}(u) \wedge talk_{*}(u)]]$.

The formulas (97) and (98) present the possible readings as far as the position of *president* with respect to the future operator is concerned. Both readings, however, say that on a moment on the future a certain person will both walk and talk. Probably this is too precise, and a better interpretation would be that there is a future interval of time in which both the walking and the talking are performed, possibly on different moments in that inverval. So this kind of objections against (97) and (98) might be solved by using another model relative to which the formulas are interpreted. But concerning the scope aspect, the formulas seem correct, and therefore the rules will produce only (97) and (98) as translations for (96).

Conjoined verb phrases with verbs which are modified differently only have a reading in which both operators have narrow scope. We have already met example (99) with reading (100).

(99) *A woman has walked and will run.*

(100) $\exists u[woman_*(u) \wedge \mathrm{H}[walk_*(u)] \wedge \mathrm{W}[run_*(u)]]$.

If the verbs of the conjoined phrase are modified in the same way, there is a reading which corresponds with the above example: sentence (101) has a reading (102)

(101) *The president will walk and will talk*

(102) $\exists u[\forall v[president_*(v) \leftrightarrow u = v] \wedge \mathrm{W}[walk_*(u)] \wedge \mathrm{W}[talk_*(u)]]$.

Sentence (101) can, however, be considered as dealing with a future president, so it also has reading (103).

(103) $\mathrm{W}\exists u[\forall v[president_*(v) \leftrightarrow u = v] \wedge walk_*(u)] \wedge talk_*(u)]$.

The possibility that the walking and talking are performed on the same moment in the future can be dealt with in the same way as I suggested for sentence (96). The fact that sentence (101) has reading (103) (= 97!) suggests us that we consider sentence (101) as a syntactic variant of (96) and assign it, too, reading (98). The same treatment will be given of the perfect.

For the past tense the same pattern applies: sentence (104) not only has reading (105) but also readings (106) and (107).

(104) *The president walked and talked.*

(105) $\exists u[\forall v[president_*(v) \leftrightarrow u = v] \wedge \mathrm{H}[walk_*(u)] \wedge \mathrm{H}[talk_*(u)]]$

(106) $H\exists u[\forall v[president_*(v) \leftrightarrow u = v] \wedge walk_*(u) \wedge talk_*(u)]$

(107) $\exists u\forall v[president_*(v) \leftrightarrow u = v] \wedge H[walk_*(u) \wedge talk_*(u)]$.

Conjoined negated verbs do not follow this pattern. Sentence (108) has reading (109), but it has no reading with only one negation sign.

(108) *The president does not walk and does not talk.*

(109) $\exists u[\forall v[president_*(v) \leftrightarrow u = v] \wedge \neg [walk_*(u)] \wedge \neg [talk_*(u)]]$.

A conjoined verb phrase which consists of equally modified verbs can, in some cases, be modified further. An example is (110), where a modifier is applied to a conjunction of verbs in the perfect.

(110) *The president will have visited Rome or have visited Tokyo.*

Conjoined verb phrases with equally modified verbs cannot be negated, as (111) illustrates. That example cannot be interpreted as a negation of a conjunction of perfect verb phrases, but only as a negated verb phrase conjoined with a non-negated one.

(111) *The president has not visited Rome or has visited Tokyo.*

If another modifier is applied first, the phrase behaves as a simple construction and can be negated, see (112).

(112) *The president will not have visited Rome or have visited Tokyo.*

5.3. Conjoined verb phrases with negated verbs

If in a conjoined verb phrase the first verb is not modified and the other verbs are negated, then the whole construction behaves as a verb phrase with unmodified verbs. This means that such a construction can be modified further; an example is (113) with reading (114).

(113) *John will walk and not talk*

(114) $W[walk_*(john) \wedge \neg [talk_*(john)]]$.

Note that sentence (113) is not ambiguous with respect to the scope of W because the interpretation of *John* is index independent. Were *John* be replaced by *the president*, then ambiguities would arise of the kind we have discussed before.

If in a conjoined verb phrase the first phrase is negated and the others are not negated, then the situation is different. A modifier 'absorbs' the negation: sentence (115) only has reading (116).

(115) *John will not walk and talk*

(116) $\neg\, W[walk_*(john) \wedge talk_*(john)]$.

If all the verbs in a conjoined verb phrase are negated, then the two patterns give rise to an ambiguity. Sentence (117) has both reading (118) and (119).

(117) *John will not walk and not talk*

(118) $W[\neg\, walk_*(john) \wedge \neg\, talk_*(john)]$

(119) $\neg W[walk_*(john) \wedge \neg talk_*(john)]$.

For conjoined verb phrases with verbs in the perfect a related situation arises. Sentences (120) and (121) seem to have one reading, whereas (122) has two readings.

(120) *John will not have walked and have talked*

(121) *John will have walked and not have talked*

(122) *John will not have walked and not have talked.*

Corresponding with the above sentences there are sentences with the contracted forms like *won't*. Sentence (123) has the same reading as its uncontracted variant (120).

(123) *John won't have walked and have talked.*

Sentence (124), however, is not equivalent with the corresponding uncontracted form (117): it only has reading (119), but not reading (118).

(124) *John won't walk and not talk.*

The way in which we may treat contracted forms depends on the organization of the morphological component. Suppose that one decides that the input of the morphological component has to consist of a string of words (where the words may bear features). Then the contraction of *will not* to *won't* cannot be dealt with in the morphological component because sentence (118) gives no syntactic information about the intended reading. This means that the contraction has to be described within the rules: the rule introducing negation should have the option of producing contracted forms like *won't*. If one has the opinion that the input of the morphological component has to be a syntactic structure, then the situation is different. I assume that sentence (117) will have a structure in which *will* is directly connected with *not* and a structure in which *walk* is directly connected with *not*. This structural information desambiguates sentence (117) and provides sufficient information to deal with contracted forms: *will not* only reduces in case it is a constituent.

5.4. Terms

The PTQ fragment only has terms which require a third person singular of the finite verb. This is probably caused by the desire to keep the syntax simple. Incorporating pronouns for the first and second person singular would not be interesting in the light of our investigations for the following reason. The pronouns *I* and *you* get an index independent interpretation and therefore (125) and (126) are not ambiguous.

(125) *I will have visited China*

(126) *You have discovered the solution.*

In what follows we will only consider 'third-person' terms.

Disjoined terms give rise to the same scope phenomena as simple terms. Sentence (127) has a reading that says that the present president or the present vice president will go, and a reading that says that the future president or future vice-president will go.

(127) *The president or the vice-president will visit Holland.*

A complication may arise from quantifying in. One might first produce (128) and obtain (127) from this by means of quantifying in.

(128) *The president or* he_1 *will visit Holland.*

This might result in a reading in which the present vice-president or the future president will visit Holland. Such mixed readings are not possible for sentence (127). This means that we have to constrain the possible applications of the quantification rule. I have not investigated these matters and I will therefore simply assume the (ad hoc) restriction that there are no terms of the form T_1 *or* T_2 in the fragment, where one or both terms are indexed pronouns.

In the examples above the determiners *the* and *a* are most frequent. For the term *every president* corresponding results are obtained: sentence (129) gets readings (130) and (131).

(129) *Every president will talk*

(130) $W[\forall u\ president_*(u) \rightarrow talk_*(u)]$

(131) $\forall u[president_*(u) \rightarrow W[talk_*(u)]]$.

If (129) concerns future presidents, it is unlikely that they have to be presidents at the same moment. One might try to represent such a meaning by formula (132).

(132) $\forall u W[president_*(u) \rightarrow talk_*(u)]$.

This is, however, not correct, since (132) would (vacuously) be true in situations such that for everyone there is a future moment at which he is *not* a president. I expect that the desired reading can be obtained by interpreting formula (130) in some model with interval semantics for time. Then (131) might get the interpretation that there is an interval in the future during which all individuals who are president in that interval will talk during that interval. The scope aspect of the meaning of (129) is then adequately represented by formulas (130) and (131).

For conjoined terms the same ambiguities will be obtained as for disjoined terms. Sentence (133) has a reading about present statesmen and one about future statesmen.

(133) *The president and the vice president will visit Cuba.*

The problem of 'mixed' readings, noticed with respect to disjunctions, also arises here, and for conjoined terms a corresponding (ad hoc) restriction on quantifying in is required. Furthermore there is the same difficulty as for the term *every president*. It is not necessary that the two statesmen of sentence (133) will visit Cuba together. A solution might be found following the suggestions concerning the interpretation of (129).

5.5. Embeddings

An important source of scope phenomena are the embedded sentences (in verb complements and in relative clauses). LADUSAW (1974) and EJERHED (1981) point out several sentences that are not treated correctly in PTQ. A variant of an example of Ladusaw is (133).

(133) *Mary has found the unicorn that walks.*

The rules produce the possible reading in which the unicorn presently walks. But they also produce a reading in which the unicorn walks on the moment of discovery (which is not a possible reading). For the future tense this ambiguity seems to be correct, see sentence (134).

(134) *Mary will find the unicorn that walks.*

An example from EJERHED (1981) is

(135) *Bill will assert that John loves Mary.*

She argues that this sentence is ambiguous. On the one reading John loves Mary at the moment of asserting, and on the other reading he loves her now. PTQ cannot distinguish between these readings, nor can the present treatment.

In order to deal with embeddings, Ladusaw makes his syntactic rules rather complex (using e.g. dominance relations) and his success is partial. I would try to find a solution in the logic. Priorian tense logic is not a suitable logical language to deal with the semantics of embedded sentences. This is illustrated by example (136).

(136) *A child was born that will become ruler of the world.*

The *will* of the embedded sentence takes as 'starting point' the reference point used for the interpretation of the whole sentence, and not the reference point introduced by the past tense. Sentence (136) was one of Kamp's arguments for introducing the 'Now'-operator (KAMP 1971). However, more power is required. The 'now'-operator keeps trace of the first point of reference one encounters during the evaluation of the sentence: the point of utterance. SAARINEN (1978) gives examples which show that one needs to be able to keep trace of all points of reference one encounters in evaluating the sentence. One of his examples is (137).

(137) *Bob mentioned that Joe has said that a child had been born who would become ruler of the world.*

Saarinen argues that the *would* can have as starting point for its evaluation any of the reference points introduced by the previous past tense operators. So each operator introduces its own variant of 'now'. This means that considerable expressive power has to be added to the logic we use for representing meanings. Since I use the logic of PTQ, with its Priorian tense operators, it is not surprising that embedded constructions in general are not treated correctly by my grammar.

6. ONE OF THE RULES

Most of the scope phenomena discussed in the previous sections will be treated explicitly: in section 7 by providing a grammar. That grammar is in some respects different from the grammar used for the PTQ fragment. The differences have already been introduced in chapter 8: words may bear features, information provided by queries is used, the rules produce labeled bracketings (or, equivalently, labeled trees), and in the formulation of the rules certain basic operations can be mentioned. These aspects of the grammar will be considered below, thereafter one of the rules will be discussed extensively.

The features are used mainly to facilitate the explicit formulation of the rules. It is, for instance, shorter to write formulation (A) instead of (B), and probably easier to understand.

(A) add features ((past, sing 3),δ)

(B) replace δ by its third person singular past tense form.

The features are not intended as a part of a general theory about features, and therefore I will only introduce those features which I find useful for the treatment of the present fragment. These are: *past*, *pc* (for participles) and *sing 3* (for the third person singular). Other features are not needed (e.g. there is no feature *pres* since $walk_{\text{sing3,past}} \rightsquigarrow$ *walked*, and $walk_{\text{sing3}} \rightsquigarrow$ *walks*).

The most important query that will be used is *Fin*. The Fins of a sentence or verb phrase are its finite verbs, i.e. the verbs which agree (in person and number) with the subject of the sentence. So it is about the same as the query Mainverb introduced in chapter 7. I prefer to avoid the name mainverb in the present context, because auxiliary verbs (such as *will* and *do*) can be used as finite verbs, and maybe not everyone would be happy to call those auxiliary verbs 'mainverbs'. The other query that will be used is *Verbphrase*. It gives the information what the verbphrase of a given sentence is. For the present fragment it turned out to be the most simple to define the queries directly on all trees, and not within the root operation (as was the method employed in chapter 7).

The labeled bracketing are used mainly to give a correct treatment of conjoined phrases. FRIEDMAN (1979) has shown that for dealing correctly with the conjoined and embedded phrases of the PTQ fragment, it is sufficient to have the bracketing available: the labels are not needed.

For the fragment under discussion the same holds: no rule needs the information provided by the labels. The choice of the labels is, for our purposes, arbitrary. Which labels actually have to be chosen, can only be decided if larger fragments of English are considered, then we might decide which rules need which information. The decision to call *will* in *John will run* an auxiliary is arbitrary from my point of view, I have no arguments pro or contra this choice. Since labels play no essential role in the discussion, I will simplify the presentation of the grammar by omitting the labels, e.g. in the presentation of the produced expressions and in the formulation of the root operations. Furthermore, I will omit brackets around simple lexical items. (e.g. using *run* instead of [*run*]). These simplifications allow me to write (139) instead of (138).

(138) $[[\mathit{John}]_T[\mathit{love}_{\mathrm{sing3}}]_{TV}[\mathit{Mary}]_T]_{IV}]_S$

(139) $[\mathit{John}[\mathit{love}_{\mathrm{sing3}}\ \mathit{Mary}]]$.

The basic operations we will use are *root* and *adjoin*. The operation *root* takes a sequence of trees as argument, and connects them with a new common root, labeled with a given category (see chapter 7). The operation *adjoin* takes two trees as argument, and connects them with a new root, which bears the same label as the root of the second argument.

As introduction to the presentation of the grammar I will extensively consider one of the rules. It is the rule which has the effect that the modifier for future tense is introduced into sentence (140), thus obtaining (141).

(140) $\mathit{John\ walk}_{\mathrm{sing3}}\ \mathit{and\ talk}_{\mathrm{sing3}}$

(141) $\mathit{John\ will}_{\mathrm{sing3}}\ \mathit{walk\ and\ talk}$.

Every sentence cannot be used as input for this rule; for instance a sentence in the future tense cannot be futurized again. There have to be restrictions on the possible applications of a rule introducing future. For other modifiers the same holds: not every sentence can be modified. One might wish to have in the grammar a single rule for the introduction on sentence level of all modifiers. This rule has to mention under which circumstances a future may be introduced, and the same for other modifiers. Moreover for each modifier it has to describe precisely in which way it has to be introduced. In this way we would obtain one great rule with a lot of subclauses. For reasons of elegancy and understandability I prefer to have for each modifier a separate rule.

F. Heny and B. Richards (eds.), Linguistic Categories: Auxiliaries and Related Puzzles Vol Two 83-99

We have to characterize the sentences to which a certain modifier can be added. There is a hierarchy which accounts for the relative scopes of modifiers as we observed this in sections 3 and 4 (conjoined phrases give rise to complications). The hierarchy is

[neg[past[fut[perf]]]].

This hierarchy claims, for instance, that negation always has wider scope than the perfect. It also says that future can be added to a positive perfect sentence and to a positive sentence in the present tense. It says moreover that future cannot be added to a negated sentence because that would give rise to an incorrect order of scope.

The hierarchy suggest to us how the possible applications of the rule introducing future has to be restricted. It can only be applied to sentences in the positive present perfect and in the positive present tense. The rule introducing future then has to contain a specification of what such sentences look like. One might expect as characterization of sentences in the positive present perfect that the sentence has as finite verb the auxiliary *have*, and as characterization of a present tensed sentence that its finite verb is in the present tense. Such a description is not sufficient. In the description of the present tensed sentences one has to exclude finite verbs which are modifiers themselves (*has,will,do*). Furthermore we have to exclude negations. If conjoined verb phrases are involved, further caution is required. These considerations show that the desired characterizations will become rather complex. I do not doubt that such characterizations are possible, but I prefer to use a somewhat different method.

The method I prefer consists of subcategorization of the sentences and verb phrases. This subcategorization is not obtained by describing explicitly which sentences belong to a certain subcategory, but indirectly by means of the rules. The rule which introduces the perfect gives the information that the sentence obtained belongs to the subcategory of sentences in the perfect tense and the rule which introduces negation gives the information that if the rule is applied to a perfect sentence the resulting sentence is the subcategory of negated perfect sentences. In this approach the rules take expressions of specified subcategories and produce expressions of specified subcategories. In this way we avoid complex conditions in the rules: the grammar does the job.

I have already mentioned two subcategories of expressions to which future tense can be added: the positive sentences in the present tense and those in the present perfect. For these subcategories I will use the names *perf S* and *S* respectively. For the category of all sentences I will use the name *full S*, so S in this chapter has a different meaning than in PTQ. In the rules many more subcategories are relevant than the ones mentioned here. The names of almost all subcategories that will be used, are indicated by the following scheme of names:

(neg)(past)(fut)(perf)S.

The names of subcategories are obtained from this scheme by replacing each subexpression of the form (α) by the expression α or by the empty string. Some examples of subcategories are as follows:

name	intuitive characterization
S	sentences in the positive present tense
neg past S	negated sentences in the past tense
past perf S	unnegated sentence in the past perfect

For verb phrases a related system of subcategories will be used. The system is somewhat larger because there are some categories for conjoined phrases, e.g. the category of conjoined phrases consisting of verbs in the perfect. The names which can be used are given by the following scheme

(conj)(neg)(past)(fut)(perf)IV.

Whether a conjoined phrase belongs to a subcategory of conjoined phrases is determined by the rules. This might have as a consequence, however, that the subcategorization of a phrase and the intuitive expectation about this do not always coincide. One might, for instance, expect that *will have walked and have talked* is a conjoined phrase. Since it behaves as a single verb in the future tense it is considered as an expression of the subcategory fut IV. For the set of all verb phrases we use the name full IV, the subcategory IV consists (in principle) of unmodified verbs.

Now I return to the rule under discussion: the one which introduces future tense. One might design a two-place rule which combines the modifier *will* with a sentence of the subcategory S or perf S. Then the rule yields a sentence of (respectively) the subcategory fut S or fut perf S. The translation of *will* introduced on the level of sentences has to be $\lambda p W[{}^{\vee}p]$, where p is a variable of type <s,t>, and the translation rule corresponding with this syntactic rule could then be $\text{MOD}'({}^{\wedge}\text{S}')$. Such a rule exhibits a a remarkable property: there is just one expression which can be used as first argument of the rule. Since only one argument is possible one could as well incorporate all information about this argument in the rule. In this way the rule with two arguments is replaced by a rule with one argument. I consider such a one-place rule as simpler and therefore I will follow this approach.

A one-place rule which introduces future in a given sentence has to contain some syntactic operation which has the effect of introducing *will*. In this way *will* becomes a syncategorematic symbol. This *will*, when considered in isolation, does not get a translation. But this does not mean that its introduction has no semantic effect: its effect is accounted for by the translation rule (which introduces the future tense operator W). Nor does the syncategorematic introduction of *will* mean that it has no syntactic status. The role of *will* in the syntax can be accounted for in the surface structure which is produced by the rule. There it can be given the position it should get on syntactic grounds and there it can get the label it should bear.

For other verb modifiers the same approach will be followed. There is no semantic or syntactic reason to have essentially different derivational histories for past sentences and sentences with future. Both verb modifiers can be introduced by means of one-place rules. That there is a great syntactic difference (in English) between past and future can be accounted for in the produced bracketing: there the introduction of past has the effect of the introduction of an affix and the introduction of future the effect of introducing an (auxiliary) verb. Also the difference between future tense in French (where it is affix) and in English can be accounted for in the labeled bracketing. Notice that the decision to introduce verb modifiers syncategorematically is not made for principled reasons, but just because it gives rise to a more elegant grammar.

Next I will consider the formulation of the rule introducing future on the level of sentences. This rule can be considered as consisting of two rules: one producing expressions of subcategory fut S (from expressions in the subcategory S) and one producing expressions of the subcategory fut perf S (from perf S expressions). The subcategorical information is combined in the following scheme (or hyperrule, see the discussion on Van Wijngaarden grammars in chapter 6, section 5):

R_{fut}: (perf)S $\rightarrow$ fut(perf)S.

From this scheme we obtain information about actual rules by replacing (perf) on both sides of the arrow consistently either by perf or by the empty string. The scheme says that there is a rule (function) from the subcategory S to the subcategory fut S and a function from the subcategory perf S to the subcategory fut perf S. Which particular rules there are is determined by the syntactic operation F_{fut}. It consists of two syntactic subfunctions which have to be performed consecutively.

F_{fut}: delete (sing 3,Fin(S));
adjoin ($will_{sing3}$,verb phrase(S)).

Agreement is dealt with in a primitive way: the rule is correct only for subjects which require the third person singular form of the verb. This is sufficient because our fragment contains only such terms. Notice that there is for both rules indicated in the scheme, one single syntactic operation. For the corresponding translation rule the same holds: there is one translation rule which reads as follows:

T_{fut}: $W[\alpha']$.

7. THE GRAMMAR

7.1. Introduction

Now we come to the kernel of the proposal: the rules. Presenting an discussion on how to treat a certain phenomenon is one step, but providing for explicit rules is another important step. The rules presented here are not just a formalization of the previous discussion. They contain more information because I have to be explicit about details I did not discuss (see also section 7.5). The rules do not deal with all phenomena mentioned in section 2 (simple modifiers) and in section 3 (compound modifiers). Furthermore the rules deal with all phenomena concerning conjoined verb phrases discussed in 4.2 and 4.3, except for the contracted forms. As for 4.4, the fragment contains disjuncted terms, but no conjuncted ones. Although embedded constructions are in the fragment, the predictions of the rules are in several cases incorrect.

The fragment described by the rules is an extension and variation of the PTQ fragment. The lexical elements are supposed to be the same as in PTQ, except for verbs like *try to*, which loose their *to*. The rules (schemes) presented below, replace the PTQ rules S_3(relative clauses), S_4(IV+T),

S_8(IV/IV+IV), S_9(S/S), S_{10}(IV//IV+IV), S_{14}(quantification of T into S), S_{15}(quantification into IV), and S_{17}(variants of S_4). Other rules are assumed to be as in PTQ, with the change that now bracketings are produced.

The rules will be presented in the form described in the previous section; i.e. by presenting their S,F, and T component. Furthermore, some of the rules are accompanied by comments or examples. In the examples the subcategory of the produced expression is mentioned between braces. The rules are divided into six groups. Each rule bears an index in the 900-series.

7.2. Rules

I. *Rules modifying verb phrases*

S_{901} : (conj)IV → perf IV
F_{901} : if *do* is among Fin(α) then delete this *do*;
add feat (pc,Fin(α)); adjoin (*have*,α)
T_{901} : $\lambda x H[\alpha'(x)]$
example: F_{901} ([*walk and*[[*do not*]*talk*]) = [*have*[$walk_{pc}$ *and*[*not* $talk_{pc}$]]] = *have walked and not talked* {perf IV}.

comment 1: The subcategory indication *conj* is not mentioned in the output subcategory because the resulting phrase behaves as an simplex verbphrase in the perfect.

S_{902} : (conj)(perf)IV → fut(perf)IV
F_{902} : if *do* occurs in Fin(α), then delete this *do*; adjoin (*will*,α)
T_{902} : $\lambda x W[\alpha'(x)]$
example: F_{902} ([*walk and*[[*do not*]*talk*]]) = [*will*[*walk and*[*not talk*]]] {fut IV}

S_{903} : (fut)(perf)IV → past(fut)(perf)IV
F_{903} : add features (past,Fin(α))
T_{903} : $\lambda x H[\alpha'(x)]$
examples: F_{903} ([*walk and*[[*do not*]*talk*]]) = [$walk_{past}$ *and*[[do_{past} *not*]*talk*]])= *walked and did not talk* {past IV}
F_{903} ([*will walk*]) = [$will_{past}$ *walk*] = *would walk* {past fut IV}

comment: Notice that this rule has the same translation rule as the rule introducing perfect (S_{901}). In case we use a logic which allows for dealing with the semantic differences between past and perfect, the translation rules would be different.

II. *Rules producing tensed sentences*

S_{904} : S → perf S
F_{904} : delete features (sing 3,Fin(α)); F_{901} (verb phrase(α));
add feat (sing 3,Fin(α))
T_{904} : H[α']
example : F_{904} ([*John walk*$_{\text{sing3}}$]) = [*John have*$_{\text{sing3}}$ *walk*$_{\text{pc}}$] =
John has walked {perf S}

comment : If one decided that *have* cannot have wide scope (see section 3), then this rule would have to be removed from the syntax.

S_{905} : (perf)S → fut(perf)S
F_{905} : delete features (sing 3,Fin(α)); F_{902} (verbphrase(α));
add features (sing 3,Fin(α))
T_{905} : W[α']
S_{906} : (fut)(perf)S → past(fut)(perf)S
F_{906} : add features (past,Fin(α))
T_{906} : H[α'].

III. *Rules for negation*

S_{907} : (conj)(past)(fut)(perf)IV → neg(past)(fut)(perf)IV
F_{907} : <u>case</u> 1 there is one verb in Fin(α):
let f be the list of features of Fin(α)
if Fin(α) is *be*, *will* or *have* then replace it by [*be*$_f$ *not*], [*will*$_f$ *not*] or [*have*$_f$ *not*] respectively;
otherwise adjoin (root(*do*$_f$,*not*),α).
<u>case</u> 2 there is more than one verb in Fin(α).
if *do* is in Fin(α) then delete this *do*;
adjoin (root(*do*,*not*),α).
T_{907} : λx ¬ [α'(x)]
examples : F_{907} ([*will walk*]) = [[*will not*]*walk*] = *will not walk* {neg fut IV}
F_{907} ([*try*[*not*[*to walk*]]]) = [[*do not*][*try*[*not*[*to walk*]]]] =
do not try not to walk {neg IV}
F_{907} ([*walk and*[[*do not*]*talk*]) = [[*do not*][*walk and*[*not talk*]]] =
do not walk and not talk {neg IV}

S_{908} : (past)(fut)(perf)S → neg(past)(fut)(perf)S
F_{908} : F_{907} (verb phrase(α))
T_{908} : ¬ α'.

IV. *IV-complements and adverbs*

S_{909} : IV//IV × (conj)(neg)(perf)IV → IV

F_{909} : if *do* is the only element of Fin(β) then produce
root(α,root(*not*,root(*to*,$\tilde{\beta}$))), where $\tilde{\beta}$ is obtained from $\tilde{\beta}$ by
deleting *do not*
otherwise
if there are occurrences of *do* in Fin(β) then delete these *do*'s;
root(α,root(*to*,β))

T_{909} : $\alpha'(^\wedge\beta')$

examples: F_{909} (*try*,[[*do not*]*run*]) = [*try*[*not*[*to run*]]] {IV}
F_{909} (*hope*,[*have talk*$_{pc}$]) = [*hope*[*to*[*have talk*$_{pc}$]]] {IV}
F_{909} (*wish*,[*walk and*[[*do not*]*talk*]])=[*wish*[*to*[*walk and*[*not talk*]]]] {IV}.

comment: The resulting phrases are of the subcategory IV because all verb modifiers can be added to them. The possible inputs of the rule are characterized as (conj)(neg)(perf)IV, predicting that all verb phrases of the corresponding categories can be input for the rule. This prediction is incorrect witness (142).

(142) *John regrets to have talked.*

Further investigations are required in order to decide which verbs take which modified complements.

S_{910} : IAV × (neg)(conj)IV → IV

F_{910} : root(β,α)

T_{910} : $\alpha'(^\wedge\beta')$

examples: F_{910} (*slowly*,*talk*) = *talk slowly*
F_{910} (*voluntarily*,[*do*[*not*[*talk*]]]) = [[*do*[*not talk*]]*voluntarily*] {IV}.

V. *Rules for conjoined phrases*

In section 5 we observed that conjoined phrases behave in various ways. This means that they are in various subcategories and that they have to be produced by several rules. The first two rules mentioned below do not create a conjoined phrase, but say that all modified verb phrases and sentences are members of the categories full IV and full S respectively. Most conjunction and disjunction rules are defined on these categories.

S_{911}	: (conj)(neg)(past)(fut)(perf)IV $\rightarrow$ full IV
F_{911}	: no change of the expression
T_{911}	: α'

S_{913}	: full IV $\times$ full IV $\rightarrow$ full IV
F_{913}	: root(α,*and* β)
T_{913}	: $\lambda x[\alpha'(x) \wedge \beta'(x)]$

S_{914}	: full S $\times$ full S $\rightarrow$ full S
F_{914}	: root(α,*and*,β)
T_{914}	: $\alpha' \wedge \beta'$

S_{915}	: as S_{913} but now for disjunction
S_{916}	: as S_{914} but not for disjunction.

The following two rules produce verb phrases which can be modified further.

S_{917}	: IV $\times$(neg)IV $\rightarrow$ conj IV
F_{917}	: root(α,*and*,β)
T_{917}	: $\lambda x[\alpha'(x) \wedge \beta'(\mathrm{x})]$

S_{918}	: as S_{917} but now for disjunction.

The following rules produce constructions with an exceptional character.

S_{919} : neg IV $\times$ neg IV $\rightarrow$ conj perf IV

F_{919} : delete *do* from α; add feature(pc,Fin(α));
delete *do* from β; add feature(pc,Fin(β));
root(*have*,root(α,*and*,β))

T_{919} : $\lambda x\mathrm{H}[\alpha'(x) \wedge \beta'(x)]$

example : S_{919} ([[*do not*]*walk*],[[*do not*]*talk*]) =
[*have*[[*not* $walk_{\mathrm{pc}}$] *and* [*not talk*]]] {conj perf IV}
The corresponding translation is
$\lambda x\mathrm{H}[walk(x) \wedge talk(x)]$.
Note that the output of S_{919} can be used as input for S_{902}, i.e.
future tense can be added to the output of S_{919}.

S_{920} : perf IV $\times$ (neg)perf IV $\rightarrow$ fut perf IV

F_{920} : delete *do* from β; adjoin (*have*,*root*(α,*and*,β))

T_{920} : $\lambda x\mathrm{W}[\alpha'(x) \wedge \beta'(x)]$

example : F_{920} (*have* $walk_{\mathrm{pc}}$,[[*do not*][*have* $talk_{\mathrm{pc}}$]]) =
[*will*[*have* $walk_{\mathrm{pc}}$] *and* [*not have* $talk_{\mathrm{pc}}$]]] =
will have walked and not have talked {fut perf IV}

S_{921} : neg perf IV × neg perf IV → neg fut perf IV
F_{921} : delete *do* from α; delete *do* from β;
adjoin(*will*,root(α,*and*,β))
T_{921} : $\lambda x W[\alpha'(x) \wedge \beta'(x)]$
example: F_{921} ([[*do not*][*have walk*$_{pc}$]],[[*do not*][*have talk*$_{pc}$]]) =
[*will*[[*not*[*have walk*$_{pc}$]] *and* [*not*[*have talk*$_{pc}$]]]] =
will not have walked and not have talked {neg fut perf IV}.

comment: If the example given with rule S_{920} is negated, the resulting phrase is identical with the example given for rule S_{921}. The respective translations are different, thus accounting for the ambiguity noted in section 5.

S_{922},S_{923},S_{924} as S_{919},S_{920},S_{921}, but now for disjunction.

VI. *Other rules*

S_{925} : T × full IV → full S
F_{925} : add feature (sing3,Fin(β))
root(α,β)
T_{925} : $\alpha'(^\wedge\beta')$

S_{926} : T × (neg)(past)(fut)(perf)IV → (neg)(past)(fut)(perf)S
F_{926} : F_{925} (α,β)
T_{926} : $\alpha'(^\wedge\beta')$

$S_{927,n}$: CN × full S → CN
$F_{927,n}$: see $F_{3,n}$ in PTQ
$T_{927,n}$: see $T_{3,n}$ in PTQ

S_{928} : S/S × full S → full S
F_{928} : adjoin (α,β)
T_{928} : see T_7 in PTQ

comment: The requirement that the sentence is an element of the category full S prevents the introduction of a verb modifier after application of S_{928}. Hence negation cannot have wide scope in:

(143) *Necessarily John does not run.*

$S_{929,n}$: T × (neg)(past)(perf)S → (neg)(past)(fut)(perf)S
$F_{929,n}$: see $F_{10,n}$ in PTQ
$T_{929,n}$: see $T_{14,n}$ in PTQ

$S_{930,n}$: T × (neg)(past)(fut)(perf)IV → (neg)(past)(fut)(perf)IV
$F_{930,n}$: see $F_{10,n}$ in PTQ
$T_{930,n}$: see $T_{15,n}$ in PTQ

S_{931},S_{932} as S_{929} and S_{930}, but now for the categories full S and full IV respectively.

7.3. Morphology

I will not explicitly describe a morphological component since that would be an ad hoc version. I have already sketched (section 4) two views on what the input for this component could be: either the whole surface structure or only the string of lexical items (with features). In both approaches it cannot be determined whether a certain occurrence of *will* was introduced on sentence level or on verb phrase level. There is for the morphological component just one *will*. Analogously there is just one *have*, whether is was introduced as auxiliary at some stage, or as a main verb describing the relation between owner and property.

7.4. Fins and verb phrase

In the rules the queries *verb phrase* and *Fin* are used. Below I will give a definition of these queries. Although I have described the framework as one which produces labeled bracketings, I did not specify labels because they are not needed in the rules. In the definition of Fin the labels are useful and I will refer to them. (If the reader has objections against this situation - not introducing the labels explicitly, but still using them - then he should neglect the labels. It does not lead to different predictions of the grammar.)

In the defintion below V is a parameter which stands for all verbal categories, i.e. V has to be replaced by IV, IV, IV//IV, or Aux (or whatever the label is of *will*, *have* and *do*). The X,Y and S stand for arbitrary labels, and ∅ for the empty set.

Fin(α)	= α	if α is a verb
Fin($[\alpha]_V$ *and* $[\beta]_V$)	= Fin(α) ∪ Fin(β)	
Fin($[\alpha]_V$ *or* $[\beta]_V$)	= Fin(α) ∪ Fin(β)	
Fin($[[\alpha]_V[\beta]_X]_V$)	= Fin(α)	
Fin($[[\alpha]_X[\beta]_S]_Y$)	= Fin(α)	if X is not a verbal category
Fin(α)	= ∅	if α does not satisfy one of the above clauses.

Verb phrase is defined analogously.

Verb phrase(α)	$= \alpha$	if α is a verb
Verb phrase($[\alpha]_S$ and $[\beta]_S$)	$=$ verb phrase (α) $\cup$ verb phrase(β)	
Verb phrase($[\alpha]_{S/S}[\beta]$)	$=$ verb phrase(β)	
Verb phrase($[[\alpha]_T[\beta]_{IV}]_S$)	$= \beta$	
Verb phrase(α)	$= \emptyset$	if α does not satisfy one of the above clauses.

7.5. Final remarks

I would like to end by saying something about the methodology. I fully agree with the final remark of PARTEE 1979a (p.94): 'It can be very frustrating to try to specify frameworks and fragments explicitly; this project has not been entirely rewarding. I would not recommend that one always work within the constraint of full explicitness. But I feel strongly that it is important to do so periodically because otherwise it is extremely easy to think that you have a solution to a problem when in fact you don't.'

Some remarks about my experiences in formulating the rules.

1. The project was not entirely successful. It was too difficult to do everything correctly at once. By providing explicit rules, I am also explicit in cases where I know the proposals to be incorrect (see section 5), or to be ad hoc (e.g. agreement).
2. The rules are explicit about borderline cases in which it is not evident that the produced sentences or the obtained readings are possible (e.g. verb phrase complements with a verb modifier).
3. The rules describe a rather large system and they make predictions about a lot of kinds of sentences I never thought of (for instance because they do not resemble the phenomena I thought of when designing the rules). I would feel safer about the correctness of the rules if I had a computer program producing hundreds of sentences of the fragment, together with their reduced translations.
4. Writing explicit rules forced me to consider the 'irrelevant' details. It turned out for instance that of the three methods for defining Fin's mentioned in JANSSEN 1980, in fact only one was applicable.
5. Considering some larger fragment explicitly gave me suggestions for finding arguments. I have presented a related treatment of verb modifiers in JANSSEN 1980 as well, but most of the arguments given in sections 2, 3 and 4 of this chapter are new, and these were found when I extended

the fragment with the quantification rules and conjunction rules.

Although the first three points are not really a recommendation for the rules presented here, I would not like to call these negative consequences of working explicitly. They are inherent to such a method of working, and constitute, in my opinion, rather an advantage. Shortcomings of a proposal with explicit rules are easier found than of a proposal without. Therefore such an approach is, generally speaking, a better starting point for further research and improvements.

APPENDIX 1

INDIVIDUAL CONCEPTS IN PTQ

In chapter 4 the syntax and semantics of the PTQ fragment were presented. The common nouns and intransitive verbs of the fragment were translated into constants which denote predicates on individual concepts. Reduction rules allowed us to replace them by predicates on individuals. What is then the benefit of using such concepts? The argument given in chapter 4 was based upon the artificial name *Bigboss*. In PTQ two less artificial examples are given as justification for the translation into predicates on individual concepts.

Consider the sentences (1) and (2).

(1) *The temperature is ninety*

(2) *The temperature is rising.*

A naive analysis of (1) and (2), using standard logic, might allow to conclude for (3).

(3) *Ninety rises.*

This would not be correct since intuitively sentence (3) does not follow from (1) and (2). So we have to provide some analysis not having this consequence. This example is known as the *temperature paradox.*

Montague's second example is a variation of the temperature paradox. Sentence (6) does not follow from sentences (4) and (5), whereas a naive analysis might implicate this.

(4) *Every price is a number*

(5) *A price rises*

(6) *A number rises.*

The solution of these problems is based upon the use of individual concepts. The idea of the solution is explained as follows. Imagine the situation that the price of oil is \$ 40, and becomes \$ 50. In this situation one might say:

(7) *The price of oil changes.*

By uttering (7) one does not intend to say that \$ 40 changes, or that \$ 50 changes. It is intended to express a property of the oil price, considered as a function from moments of time to amounts of dollars. Therefore (7) could be translated into a formula expressing a property of an individual concept: the oil price concept. Formally spoken, prices are considered as functions from indices to numbers, and the same for temperatures. Numbers are considered as elements in D_e, so prices and temperatures are of type <s,e> they are individual concepts.

The technical details of the solution of the temperature paradox can be illustrated by the treatment of sentence (1). The first step of its production is application of S_5 to *be* and *ninety*. This yields (8); the corresponding translation reduces to (9).

(8) *be ninety*

(9) $\lambda x[{}^{\vee}x = ninety]$.

The next step is to combine (8) with term (10), which has (11) as translation.

(10) *the temperature*

(11) $\lambda P[\exists x \forall y[temperature(y) \leftrightarrow x = y] \wedge {}^{\vee}P(x)]$.

Since the meaning postulate for common nouns (MP2) does not hold for *temperature*, its translation (11) cannot be reduced to a formula with quantification over individuals. Combination of (8) with (11) according to S_4 yields sentence (1); the corresponding translation reduces to (12).

(12) $\exists x[\forall y[temperature(y) \leftrightarrow x = y] \wedge {}^{\vee}x = ninety]$.

The translation of sentences (2) and (3) are respectively (13) and (14).

(13) $\exists x[\forall y[temperature(y) \leftrightarrow x = y] \wedge rise(x)]$

(14) $rise({}^{\wedge}ninety)$.

From (12) and (13) it does not follow that (14) is true.

Montague's treatment of the temperature paradox has been criticized for his analysis of the notion temperature. But there are examples of the same phenomenon which are not based upon temperatures (or prices). Several examples are given by LINK (1979) and LOEBNER (1976). One of their examples is the German version of (15).

(15) *The trainer changes.*

On the reading that a certain club gets another trainer, it would not be correct to translate (15) by a formula which states that the property of changing holds for a certain individual.

The temperature paradox (and related phenomena) explain why individual concepts are useful. But in most circumstances we want to reduce them to individuals. In the remainder of this appendix it will be investigated when such a reduction is allowed. First we will do so for translations of intransitive verbs, then for other verbs, and finally for translation of common nouns.

The only intransitive verbs in the PTQ fragment which do not express a property of an individual, but of an individual concept, are *rise* and *change*. Therefore we restrict our attention to those models of IL in which the constants corresponding with the other intransitive verbs are interpreted as expressing properties of individuals. This is expressed by the following meaning postulate.

1. <u>Meaning Postulate 3</u>

$$\exists M \forall x \Box [\delta(x) \leftrightarrow [{}^{\vee}M]({}^{\vee}x)] \quad (M \in VAR_{<s,<e,t>>})$$

where $M \in VAR_{<s,<e,t>>}$ and δ translates any member of B_{IV} other than *rise* or *change*.

1. END

This meaning postulate states that for all involved predicates on individual concepts there is for each index an equivalent predicate on individuals. This predicate is index dependent: the set of walkers now may differ from the set of walkers yesterday. MP3 expresses the existence of such an equivalent predicate by the existential quantification $\exists M$. This M is of type $<s,<e,t>>$ because variables get an index independent interpretation, and as argued before, the predicate on individuals corresponding with δ has to be index dependent.

In chapter 4 section 2, the δ_* notation was introduced as an abbreviation for $\lambda u[\delta({}^{\wedge}u)]$, so as an abbreviation for those cases where it could be said that δ was applied to an individual. The above meaning postulate says that for certain constants δ the argument always is an individual, even if this is not appearent from the formula. Therefore it might be expected that MP3 allows us to introduce the δ_* notation for those constants in all contexts. The following theorems allow us to replace a formula with an occurrence of δ (where MP3 holds for δ), by a formula with an occurrence of δ_*.

2. <u>THEOREM</u>. MP3 *is equivalent with*

$$\models \Box\, \delta(x) \leftrightarrow \delta_*({}^\vee x).$$

<u>PROOF part 1</u>. Suppose that MP3 holds, so $\models \exists M \forall x \Box\, [\delta(x) \leftrightarrow [{}^\vee M]({}^\vee x)]$.
Then there is a g such that $g \models \forall x \Box\, [\delta(x) \leftrightarrow [{}^\vee M]({}^\vee x)]$.
Now for all $g' \underset{v}{\sim} g$ $g' \models \delta_*(v) = \lambda u[\delta({}^\wedge u)](v) = \delta({}^\wedge v) = [{}^\vee M]({}^{\vee\wedge} v) = [{}^\vee M](v)$.
Consequently $g' \models \delta_* = {}^\vee M$.
So there is a g: $g \models \forall x \Box\, [\delta(x) \leftrightarrow \delta_*({}^\vee x)]$.
Since there are no free variables in this formula, we have

$$\models \Box\, [\delta(x) \leftrightarrow \delta_*({}^\vee x)].$$

<u>REMARK</u>. The following more direct approach is incorrect because the conditions for λ-conversion are not satisfied.

$$g \models \delta_*({}^\vee x) = \lambda u \delta({}^\wedge u)({}^\vee x) = \delta({}^{\wedge\vee} x) = \delta(x) = [{}^\vee M]({}^\vee x).$$

<u>PROOF part 2</u>. Suppose $\models \Box\, [\delta(x) \leftrightarrow \delta_*({}^\vee x)]$.
Let g,i be arbitrary and define $g' \underset{M}{\sim} g$ by $g'(M) = [{}^\wedge \lambda u \delta({}^\wedge u)]^{A,i,g}$.
Then $i,g' \models \delta(x) \leftrightarrow [\lambda u \delta({}^\wedge u)]({}^\vee x) \leftrightarrow [{}^{\vee\wedge} \lambda u \delta({}^\vee x)] \leftrightarrow [{}^\vee M]({}^\vee x)$.
Since g,i were arbitrary, MP2 follows.
2. END

On the basis of this theorem, we have besides RR_3, another reduction rule introducing the *.

3. <u>Reduction rule 11</u>

Let be given an expression of the form $\delta(x)$, where δ is the translation of an intransitive verb other than *rise* or *change*.
Then replace $\delta(x)$ by $\delta_*({}^\vee x)$.

CORRECTNESS PROOF

Apply theorem 2.
3. END

Now we have two rules for the introduction of δ_*: RR_3 and RR_{11}. The one requires that the argument is of a certain form, the other that the function is of a certain nature. They have different conditions for application, and none makes the other superfluous. In case both reduction rules are applicable, they yield the same result. It is not clear to me why MP3 is formulated as it is, and not directly in the form given in theorem 2.

For verbs of other categories there are related meaning postulates. For instance the transitive verb *find* should be interpreted as a relation between individuals. The meaning postulate for the transitive verbs were already given in chapter 4 (MP4). Exceptions to that meaning postulate were *seek* and *conceive* because these verbs do not express a relation between individuals. But also about these verbs something can be said in this respect. The first arguments have to be (intensions of) individuals: it is an individual that seeks, and not an individual concept. This is expressed by meaning postulate 5, that will be given below. For verbs of other categories a related postulate expresses that their subjects are not individual concepts, but individuals.

4. Meaning postulate 5

$$\forall P\exists M\forall x\Box[\delta(x,P) \leftrightarrow [{}^{\vee}M]({}^{\vee}x)]$$

where $\delta \in$ {*seek, conceive*}.

5. Meaning postulate 6

$$\forall p\exists M\forall x\Box[\delta(x,p) \leftrightarrow [{}^{\vee}M]({}^{\vee}x)]$$

where $\delta \in$ {*believe that, assert that*}.

6. Meaning postulate 7

$$\forall P\exists M\forall x\Box[\delta(x,P) \leftrightarrow [{}^{\vee}M]({}^{\vee}x)]$$

where $\delta \in$ {*try to, wish to*}.
6. END

These three meaning postulates do not give rise to new reduction rules because there are no generally accepted notations for the corresponding predicates with an individual as first argument.

The treatment of the temperature paradox was essentially based on the use of individual concepts.

This explains why all common nouns are translated into constants denoting predicates on individual concepts. Most common nouns express a predicate on individuals. This is formulated in a meaning postulate which I recall from chapter 4.

7. Meaning postulate 2

$$\Box[\delta(x) \rightarrow \exists u[x = {}^{\wedge}u]]$$

where $\delta \in \{man, woman, park, fish, pen, unicorn\}$.
7. END

The meaning postulates for nouns and for verbs have a related aim: they both aim at excluding arbitrary individual concepts as argument and guaranteeing an individual as argument. So one might expect that there is a close relation between the consequences of the two meaning postulates. One might for instance expect that for nouns something holds like the formula in MP3. This is not the case, as is expressed in the following theorem.

8. THEOREM. *Let* $\delta \in CON_{<<s,e>,t>}$.
Let (I) *be the formula* $\Box[\delta(x) \rightarrow \exists u[x={}^{\wedge}u]]$
and (II) *the formula* $\Box[\delta(x) \leftrightarrow \delta_{*}({}^{\vee}x)]$.
Then (i) (I) $\not\models$ (II)
and (ii) (II) $\not\models$ (I).

PROOF. (i) In chapter 3 we introduced the constant *bigboss*, which will be used here. Suppose that

$$i_1 \models bigboss = {}^{\wedge}nixon \text{ and } i_2 \models bigboss = {}^{\wedge}bresjnev.$$

Then

$$[{}^{\vee\wedge}bigboss]^{A,i_1,g}(i_2) = \lambda i[bigboss^{A,i_1,g}(i)](i_2) = bigboss^{A,i_2,g}(i_2) = bresjnev$$

and

$$[{}^{\wedge\vee}bigboss]^{A,i_1,g}(i_1) = \lambda i[bigboss^{A,i_1,g}(i)](i_1) = bigboss^{A,i_1,g}(i_1) = nixon.$$

This means that ${}^{\wedge\vee}bigboss$ is an expression of type <s,e> which does not denote a constant function. Since $[{}^{\wedge}u]^{A,i_1,g}(i_2) = [{}^{\wedge}u]^{A,i_1,g}(i_1) = \underline{\lambda}_i g(u)$

we have that for no g: $g,i_1 \models {}^{\wedge}u = {}^{\wedge\vee}bigboss$. Suppose furthermore that δ is a constant for which MP_2 holds, say *man*.
Then (I) is satisfied.
So $g,i_1 \models man({}^{\wedge\vee}bigboss) \rightarrow \exists u[{}^{\wedge\vee}bigboss = {}^{\wedge}u]$.
Due to the just proved property of ${}^{\wedge\vee}bigboss$, the consequence is never true.
So for no g $g,i_1 \models man({}^{\wedge\vee}bigboss)$.
Suppose moreover that the predicate man_* holds for Nixon.
So $g,i_1 \models man_*(nixon)$.
Since $g,i_1 \models {}^{\vee}bigboss = nixon$
we have $g,i_1 \models man_*({}^{\vee}bigboss)$. Consequently
for no g $g,i_1 \models man({}^{\vee\wedge}bigboss) \leftrightarrow man_*({}^{\vee}bigboss)$.
So if $g(x) = [{}^{\wedge\vee}bigboss]^{A,i_1,g}$ statement (II) is not true. Finally, note that it is easy to design a model in which *bigboss* and *man* have the assumed properties. Hence we have proven (i).

PROOF. (ii) Let *bigboss* be as above. Assume now that δ is a constant for which MP_3 holds, say *walk*. Then (II) holds for δ.
Suppose now $i_1 \models walk_*(nixon)$.
So $i_2 \models walk_*({}^{\vee}bigboss)$.
Let $g(x) = [{}^{\wedge\vee}bigboss]^{A,i,g}$.
Then it is not true that

$$i \models walk_*(x) \rightarrow \exists u[x={}^{\wedge}u]$$

because the antecedence is true, whereas the consequence is false. A model in which *walk* and *bigboss* have the desired properties can easily be defined and that is a counterexample to the implication.
8. END

Consequences of the above theorem are:
1. The formulations of the meaning postulates for Common Nouns and for Intransitive Verbs cannot be transformed into each other.
2. The following statement from PTQ (MONTAGUE 1973, p.265,+19) is incorrect: $\Box[\delta(x) \leftrightarrow \delta_*({}^{\vee}x)]$ if δ translates a basic common noun other than *price* or *temperature*.
3. The meaning postulate for common nouns does not allow for replacing in all contexts an individual concept variable by the extension of this variable. This result was independently found by LINK (1979, p.224).

Next I will prove that in certain contexts the meaning postulate for common nouns does allow us to replace bound variables of type <s,e> by variables of type e. It are contexts created by these translation rules of the PTQ-fragment: the translation rule for the determiner CN and the determiner-CN-rel.clause constructions. In the sequel δ stands for the translation of a CN for which MP_2 holds, and ϕ for the translation of a relative clause. This ϕ may be omitted in the formulation of the theorems.

9. LEMMA. *If* $A,i,g \models \forall x\psi$ *then* $A,i,g \models \forall u[{}^{\wedge}u/x]\psi$
if $A,i,g \models \exists u[{}^{\wedge}u/x]\psi$ *then* $A,i,g \models \exists x\psi$.

PROOF. $\{m \in D_{<s,e>} | m = [{}^{\wedge}u]^{A,i,g}\} \subset \{m \in D_{<s,e>} | m = x^{A,i,g}\}$.
9. END

The next theorem deals with terms in which the determiner is *a*.

10. THEOREM. $A,i,g \models \exists x[\delta(x) \wedge \phi \wedge {}^{\vee}P(x)]$
iff $A,i,g \models \exists u[\delta({}^{\wedge}u) \wedge [{}^{\wedge}u/x]\phi \wedge {}^{\vee}P({}^{\wedge}u)]$.

PROOF. Suppose that

(1) $$A,i,g \models \exists x[\delta(x) \wedge \phi \wedge {}^{\vee}P(x)].$$

Then there is a m $\in D_{<s,e>}$ such that

(2) $$A,i,[x{\to}m]g \models \delta(x) \wedge \phi \wedge {}^{\vee}P(x).$$

From MP_2 and (2) follows

(3) $$A,i,[x{\to}m]g \models \exists u[x={}^{\wedge}u].$$

So there is a a $\in D_e$ such that

(4) $$A,i,[x{\to}m,u{\to}a]g \models x = {}^{\wedge}u.$$

From (4) and (2) follows

(5) $$A,i,[x{\to}m,u{\to}a]g \models \delta({}^{\wedge}u) \wedge [{}^{\wedge}u/x]\phi \wedge {}^{\vee}P({}^{\wedge}u).$$

So

(6) $\quad A,i,g \models \exists u[\delta(^{\wedge}u) \wedge [^{\wedge}u/x]\phi \wedge {}^{\vee}P(^{\wedge}u)].$

Reversely (6) implies (1), as follows from the above lemma.
10. END

The terms with determiner *every* are dealt with in theorem 11.

11. THEOREM. $A,i,g \models \forall x[\delta(x) \wedge \phi \rightarrow {}^{\vee}P(x)]$
iff $A,i,g \models \forall u[\delta(^{\wedge}u) \wedge [^{\wedge}u/x]\phi \rightarrow {}^{\vee}P(^{\wedge}u)].$

PROOF. One direction of the theorem follows immediately from Lemma 9. The other direction is proved by contra-position. Assume that was *not* true that

(1) $\quad A,i,g \models \forall x[\delta(x) \wedge \phi \rightarrow {}^{\vee}P(x)].$

This means

(2) $\quad A,i,g \models \neg \forall x[\delta(x) \wedge \phi \rightarrow {}^{\vee}P(x)].$

This is equivalent with

(3) $\quad A,i,g \models \exists x[\delta(x) \wedge \phi \wedge \neg {}^{\vee}P(x)].$

Application of the argumentation of theorem 10 gives

(4) $\quad A,i,g \models \exists u[\delta(^{\wedge}u) \wedge [^{\wedge}u/x]\phi \wedge \neg {}^{\vee}P(^{\wedge}u)].$

Therefore it is not true that

(5) $\quad A,i,g \models \forall u[\delta(^{\wedge}u) \wedge [^{\wedge}u/x]\phi \rightarrow {}^{\vee}P(^{\wedge}u)].$

So (5) implicates (1).
11. END

The next two theorems deal with terms with determiner *the*.

12. <u>THEOREM</u>. *If* A,i,g $\models \exists y[\forall x[[\delta(x) \wedge \phi] \leftrightarrow x=y] \wedge {}^{\vee}P(y)]$.
Then A,i,g $\models \exists u[\forall v[[\delta({}^{\wedge}v) \wedge [{}^{\wedge}v/x]\phi] \leftrightarrow u=v] \wedge {}^{\vee}P({}^{\wedge}u)]$.

<u>PROOF</u>. Suppose

(1) $\qquad$ $A,i,g \models \exists y[\forall x[\delta(x) \wedge \phi \leftrightarrow x=y] \wedge {}^{\vee}P(y)]$.

This means that there is an $m \in D_{<s,e>}$ such that (2) and (3) hold

(2) $\qquad$ $A,i,[y\to m]g \models \forall x[\delta(x) \wedge \phi \leftrightarrow x=y]$

(3) $\qquad$ $A,i,[y\to m]g \models {}^{\vee}P(y)$.

From (2) follows (4), and therefore (5) holds.

(4) $\qquad$ $A,i,[y\to m]g \models \delta(y) \wedge [y/x]\phi \leftrightarrow y = y$

(5) $\qquad$ $A,i,[y\to m]g \models \delta(y) \wedge [y/x]\phi$.

From (5) and MP_2 follows that there is an $a \in D_e$ such that (6)

(6) $\qquad$ $A,i,[y\to m,u\to a]g \models y = {}^{\wedge}u$.

From (3) and (6) follows (7)

(7) $\qquad$ $A,i,[y\to m,u\to a]g \models {}^{\vee}P({}^{\wedge}u)$.

Apply lemma 9 to (2) and substitute ${}^{\wedge}v$ for y. Since (6) holds it follows that (8) holds

(8) $\qquad$ $A,i,[y\to m,u\to a] \models \forall v[\delta({}^{\wedge}v) \wedge [{}^{\wedge}v/y]\phi \leftrightarrow {}^{\wedge}v={}^{\wedge}u]$.

Since $[u=v]^{A,i,g}$ equals $[{}^{\wedge}u={}^{\wedge}v]^{A,i,g}$, we may replace in (8) ${}^{\wedge}v = {}^{\wedge}u$ by $v = u$. Combination of (8) with (7) yields (9)

(9) $\qquad$ $A,i,[y\to m,u\to a] \models \forall v[\delta({}^{\wedge}v) \wedge [{}^{\wedge}v/y]\phi \leftrightarrow v = u] \wedge {}^{\vee}P({}^{\wedge}u)]$.

From this the theorem follows.

12. END

13. <u>THEOREM</u>. *If* $A,i,g \models \exists v[\forall u[[\delta({}^{\wedge}u) \wedge [{}^{\wedge}u/x]\phi] \leftrightarrow u=v] \wedge {}^{\vee}P({}^{\wedge}v)]$ *then* $A,i,g \models \exists y[\forall x[[\delta(x) \wedge \phi] \leftrightarrow x=y] \wedge {}^{\vee}P(x)]$.

<u>PROOF</u>. Suppose

(1) $$A,i,g \models \exists v[\forall u[\delta({}^{\wedge}u) \wedge [{}^{\wedge}u/x]\phi \leftrightarrow u=v] \wedge {}^{\vee}P({}^{\wedge}v)].$$

Then there is an $a \in D_e$ such that (2) and (3) hold

(2) $$A,i,[v{\rightarrow}a]g \models \forall u[\delta({}^{\wedge}u) \wedge [{}^{\wedge}u/x]\phi \leftrightarrow u=v]]$$

(3) $$A,i,[v{\rightarrow}a]g \models {}^{\vee}P({}^{\wedge}v).$$

Let $m \in D_{<s,e>}$ be such that (4) holds.

(4) $$A,i,[x{\rightarrow}m]g \models \delta(x) \wedge \phi.$$

Then from MP_2 follows that there is a b such that

(5) $$A,i,[x{\rightarrow}m,u{\rightarrow}b]g \models x = {}^{\wedge}u.$$

From (5) and (2) follows (6)

(6) $$A,i,[x{\rightarrow}m,v{\rightarrow}a,u{\rightarrow}b]g \models \delta(x) \wedge \phi \leftrightarrow x = {}^{\wedge}v.$$

Since (4) holds, it follows from (6).

(7) $$A,i,[x{\rightarrow}m,v{\rightarrow}a]g \models x = {}^{\wedge}v.$$

It follows from (4) and (7) that (8) holds

(8) $$A,i,[v{\rightarrow}a]g \models \forall x[\delta(x) \wedge \phi \rightarrow x={}^{\wedge}v].$$

Let now $m \in D_{<s,e>}$ be such that (9) holds

(9) $A,i,[v\to a,x\to m]g \models x = {}^{\wedge}v.$

From (2) it then follows that (10) holds

(10) $A,i,[v\to a,x\to m]g \models \delta(x) \wedge \phi.$

From (9) and (10) follows (11)

(11) $A,i,[v\to a]g \models \forall x[x={}^{\wedge}v \to \delta(x) \wedge \phi].$

From (3), (8) and (11) the theorem follows.
13. END

The above theorems constitute the justification for the following reduction rule.

14. REDUCTION RULE 12

Let δ be the translation of a common noun for which meaning postulate MP_2 holds. Let be given a formula of one of the following forms,

$$\exists x[\delta(x) \wedge \phi \wedge {}^{\vee}P(x)]$$

$$\forall x[\delta(x) \wedge \phi \to {}^{\vee}P(x)]$$

$$\exists y[\forall x[\delta(x) \wedge \phi \leftrightarrow x=y] \wedge {}^{\vee}P(y)].$$

Then replace this formula by respectively

$$\exists u[\delta({}^{\wedge}u) \wedge [{}^{\wedge}u/x]\phi \wedge {}^{\vee}P({}^{\wedge}u)]$$

$$\forall u[\delta({}^{\wedge}u) \wedge [{}^{\wedge}u/x]\phi \to {}^{\vee}P({}^{\wedge}u)]$$

$$\exists v[\forall u[\delta({}^{\wedge}u) \wedge [{}^{\wedge}u/x]\phi \leftrightarrow u=v] \wedge {}^{\vee}P({}^{\wedge}v)]$$

(provided that ϕ does not contain a free occurrence of u or v).

CORRECTNESS PROOF

See the theorems.

14. END

The theorems mentioned above, allow us to change the types of bound variables in a lot of contexts which arise if one deals with sentences from the PTQ-fragment. But they do *not* cover all contexts arising in this fragment. If the rule of quantification into a CN phrase (i.e. $S_{15,n}$) is used, then no reduction rule is applicable. An example is (14) in the reading in which *every* has wider scope than *a*. The corresponding translation is (15), and although none of the reduction rules is applicable, it is equivalent with (16).

(14) *Every man such that he looses a pen such that he finds it, runs.*

(15) $\forall x[\exists u[pen_*(u) \wedge man(x) \wedge loose_*({}^{\vee}x,u) \wedge find_*({}^{\vee}x,u)] \rightarrow run_*({}^{\vee}x)]$

(16) $\forall v[\exists u[pen_*(u) \wedge man_*(v) \wedge loose_*(v,u) \wedge find_*(v,u)] \rightarrow run_*(v)]$.

One would like to have a reduction rule which is applicable to constructions in which quantification into a CN is used. However, not in all such contexts reduction is possible. This was discovered by FRIEDMAN & WARREN (1980a). Consider sentence (17)

(17) *A unicorn such that every woman loves it changes.*

Suppose that 17 is obtained by quantification of *every woman* into *unicorn such that he$_1$ loves it.* Then the translation of (17) reduces to (18); Friedman & Warren call this 'a rather unusual reading'.

(18) $\exists x[\forall u[woman_*(u) \rightarrow unicorn(x) \wedge love_*(u,{}^{\vee}x) \wedge change(x)]]$.

This translation is, however not equivalent with (19).

(19) $\exists v[\forall u[woman_*(u) \rightarrow unicorn_*(v) \wedge love_*(u,v) \wedge change_*(v)]]$.

This situation might rise doubts about rule $S_{15,n}$, However, see chapter 9, section 7.2 for an example where the rule is needed.

APPENDIX 2

SET MANIPULATION IN SYNTAX

In chapter 6 I provided a system of categories for dealing with syntactic variables. The rules given there implicitly assume that the reader knows what sets are, and what $\cup$, with and - mean. This is set theoretical knowledge, and not knowledge of the grammatical system. In the present section I will formulate syntactic rules which allow for replacing expressions like $\{1,2\} - 1$ by $\{2\}$ and $\{1,2\} \cup 3$ by $\{1,2,3\}$. So we aim at rules which remove the symbols $\cup$, with and - from the formulation of the rules. The collection of rules performing this task is rather complex. I wish to emphasize that the rules do not arise from the requirement of using total rules, but from grammatical formalism. A related situation would arise when using partial rules. Such rules would mention a condition like 'contains an occurrence of he_n'. Since 'containing' is not a notion defined by grammatical means, a formalist might wish to do so. Then rules are needed which are related to the rules below since they have to perform related tasks. Once it is shown that the set-theoretical notions $\cup$, with and - are definable by means of grammatical tools, there is no objection against using them in the grammar even when not explicitly defined in this way.

Let G be a grammar with a collection hyperrules H, and let the elements of H contain expressions like *set* - *n*. Then the actual rules of the grammar are defined as the result of performing the following actions in order.

1. replace the metavariables in the hyperrules by some terminal metaproduction of the meta-grammar
2. replace subexpressions in the rules by other expressions, according to the rules given below, until there are no occurrences of the non-acceptable symbols (U, with, -).

The rules eliminating the non acceptable symbols introduce some non-acceptable symbols themselves. These are +, is, unless, true and false; these symbols have to be added to those mentioned in point 2 above. The collection of rules performing the task of eliminating these symbols is infinite, and will be defined by means of a two-level grammar. The hyperrules describing the elimination of the unacceptable symbols are unrestricted rewriting rules with metavariables. These variables are mentioned below, together with some examples of their terminal productions. Different examples are separated by a bar symbol: /, and ε denotes the empty string.

set : {1,2} / {3,1} / ∅.
seq : 1,2 / 3,1 .
lseq: 1, / 3,1, / ε.
rseq: ,2 / ,1,5 / ε.
n : 1 / 5.

The metarules for these metavariables are as follows (again a bar / separates alternatives, the non-terminal symbols are in italics).

set → *seq* / ∅.
seq → *n* / *n*, *seq*.
n → 1/2/3/4/5/6/7/8/9/*nn*/*n*0.
lseq → *seq*, / ε.
rseq → ,*seq* / ε.

The rules for with have to allow for replacing {1,2} with 1 by {1,2}, whereas they should not allow for replacing {2,3} with 1 by {2,3}. The hyperrule describing such replacements is

{*lseq*,*n*,*rseq*} with *n* → {*lseq*,*n*,*rseq*}.

An example of a rule derived from this hyperrule is

{1,3,5} with 3 → {1,3,5}.

Thus the expression with 3 is eliminated. In case one meets the subexpression {2,3} with 1 there is no rule which can be obtained from this hyperrule and which can be applied to this subexpression. So we cannot get rid of the non-acceptable symbol with, as was required in point 2. So we do not obtain an actual rule and the derivation cannot continue. This 'blind alley' technique is due to SINTZOFF (1967).

The rules for eliminating the - sign have to replace {1,2} - 2 by {1}, and {1,2} - 3 by {1,2}. The rules have to check whether the number preceeded by the - sign occurs in the set mentioned before the sign. For this purpose, we need grammatical means to check whether two numbers are equal or different. It is easy to design a rule which can be applied only if two numbers are equal: a hyperrule with two occurrences of the meta-variable *n* can be transformed into a real rule only by substituting for both occurrences the same number. If a hyperrule contains metavariables n_1 and n_2, then it can be transformed into a real rule by substituting for n_1 and n_2 different numbers. But nothing prevents us to substitute the same number. It is difficult to guarantee that two numbers are different, but we need such rules. The rules which do so use the blind alley technique again, now on the symbol $\underline{\text{unless}}$. The hyperrules are as follows.

$$\begin{array}{lll}
0\ \underline{\text{is}}\ 0 \to \underline{\text{true}} & 1\ \underline{\text{is}}\ 0 \to \underline{\text{false}} & 2\ \underline{\text{is}}\ 0 \to \underline{\text{false}} \\
0\ \underline{\text{is}}\ 1 \to \underline{\text{false}} & 1\ \underline{\text{is}}\ 1 \to \underline{\text{true}} & 2\ \underline{\text{is}}\ 1 \to \underline{\text{false}} \\
\vdots & \vdots & \vdots \\
0\ \underline{\text{is}}\ 9 \to \underline{\text{false}} & 1\ \underline{\text{is}}\ 9 \to \underline{\text{false}} & \\
1n_1\ \underline{\text{is}}\ 1n_2 \to n_1\ \underline{\text{is}}\ n_2 & & 2n_1\ \underline{\text{is}}\ 1n_2 \to \underline{\text{false}} \\
1n_1\ \underline{\text{is}}\ 2n_2 \to \underline{\text{false}} & & 2n_1\ \underline{\text{is}}\ 2n_2 \to n_1\ \underline{\text{is}}\ n_2 \\
\vdots & & \\
1n_1\ \underline{\text{is}}\ 9n_2 \to \underline{\text{false}} & & \\
\underline{\text{unless}}\ \underline{\text{true}} \to \underline{\text{false}} & & \\
\underline{\text{unless}}\ \underline{\text{false}} \to \underline{\text{true}} & & \\
\underline{\text{true}} \to \varepsilon. & &
\end{array}$$

The above rules have the effect that an expression of the form $\underline{\text{unless}}$ a $\underline{\text{is}}$ b reduces to $\underline{\text{unless}}\ \underline{\text{true}}$ and next to $\underline{\text{unless}}$ in case a equals b. This $\underline{\text{unless}}$ constitutes a blind alley. If a is not equal to b, the expression reduces to $\underline{\text{unless}}\ \underline{\text{false}}$ and through $\underline{\text{true}}$ to the empty string. Then the test is eliminated, and the production may proceed.

The rules for the - sign have to check for all elements of the mentioned set whether they are equal to the element that has to be removed. The element for which equality is tested (in a step of the testing process) is the last element of the sequence describing the set. If equality is found, the element is removed. If a check shows that the numbers are different, then the element which has been checked, is put at the beginnings of the sequence, and the new 'last element' is checked. By means of the * sign the numbers are separated which are already checked from the numbers which are not yet checked. This rotation technique is due to Van WIJNGAARDEN (1974). The hyperrules introducing and removing the * sign are as follows.

$\{seq\} - n \rightarrow \{*, seq\} - n$

$\emptyset - n \rightarrow \emptyset$

$\{seq, *\} - n \rightarrow \{seq\}$

$\{*\} - n \rightarrow \emptyset.$

The hyperrule removing an element is

$\{lseq * reseq, n\} - n \rightarrow \{lseq * rseq\} - n.$

The rule rotating the sequence is

$\{lseq * rseq, n_1\} - n_2 \rightarrow \{n_1, lseq * rseq\} - n_2$ <u>unless</u> n_1 <u>is</u> n_2.

We use the <u>unless</u> phrase to guarantee that n_1 and n_2 are different. If the numbers are different, then the phrase reduces to the empty string. If they are equal the <u>unless</u> phrase reduces to the expression <u>unless</u>, and we cannot get rid of this phrase. This means that we are in a blind alley: we do not get an actual rule.

The rules for ∪ use the - sign. It would be easy to reduce {1,2} ∪ {2} to {1,2,2} but, in order to avoid this repetition of elements, I first remove the 2 from the leftmost set and then add 2 to the set thus obtained. The rules are as follows.

$set \cup \{n, rseq\} \rightarrow set - \mathrm{n} + \mathrm{n} \cup \{rseq\}$

$\{seq\} + n \rightarrow \{seq, n\}$

$\emptyset + n \rightarrow n$

$set \cup \{\emptyset\} \rightarrow set$

$set \cup \{\ \} \rightarrow set.$

This completes the description of the set of rules needed for dealing with set-theoretical notions by grammatical means.

INDEX OF NAMES

Andreka 87
Bach 119, 120, 122, 124, 125, 133, 138, 144-147, 150, 151, 157, 164, 167, 168, 178, 180-183
Bartsch 43, 54, 61, 95
Bennett 129, 154, 177, 178
van Benthem 11, 77
Bresnan 112, 141
Burmeister 87
Chomsky 102, 103, 131, 138
Cooper 54, 86, 87, 119, 120, 137, 138, 141, 144-158, 166, 167
Curry 122
Delacruz 57
Dik 118
Dowty 43, 57, 69, 95, 122, 124, 177
Ejerhed 192, 193,
Emde Boas 29, 75
Faltz 71, 72, 77, 78
van Fraassen 86
Friedman 7, 22, 24, 37, 118, 131, 132, 167, 221
Gallin 74, 76
Gazdar 21, 103
Goguen 87
Goldblatt 87
Graetzer 87
Groenendijk 70, 71, 76, 124, 128, 143, 150, 158
Hausser 75, 76, 86, 87, 111, 124, 141, 156
Henkin 74
Hopcroft 84
Hopkins 184
Indurkhya 7
Janssen 7, 27, 54, 75, 89, 94, 118, 138 144, 155, 157, 178, 206
Kamp 87, 193
Karttunen 166
Keenan 71, 72, 77
Klein 87, 131
von Kutschera 87
Kripke 5
Ladusaw 192
Landman 33, 121, 125
Landsbergen 107, 122
Lewis 10, 48, 76
Link 210
Loebner 210
Lukasiewics 87
ter Meulen 87
Mikenberg 87
Moerdijk 121, 125
Montague 1, 2, 4, 58, 62, 78, 124, 150, 174, 215

Nemeti 87
Oh 158
Parsons 54, 119
Partee 25, 27, 49, 56, 57, 73, 80, 82, 91-99, 108-112, 117-120, 125, 129, 131, 132, 138, 140, 141, 155, 156, 163, 164, 206
Rodman 60, 141, 143, 147, 157
Rohrer 177,
Saarinen 193
Sag 103
Scha 85
Scott 87
Sintzoff 224
Stokhof 70, 71, 76, 124, 128, 143, 150, 158, 176
Strachey 87
Thomason 63, 82, 84, 94, 143, 155, 159
Tichy 75
Veltman 91
Waldo 85, 87
Warren 7, 37, 167, 221
van Wijngaarden 103, 225

REFERENCES

Andreka, H., P. Burmeister & I. Nemeti, 1980,
'Quasivarieties of partial algebras. A unifying approach towards a two-values model theory for partial algebras',
Preprint 557, Fachbereich Mathematics, Technische Hochschule, Darmstadt.

Andreka, H., & I. Nemeti, 1982,
'Generalization of variety and quasi variety-concept to partial algebras through category theory',
Dissertationes Mathematicae 204.

Bach, E. & R.H. Cooper, 1978,
'The NP-S analysis of relative clauses and compositional semantics',
Linguistics and Philosophy 2, 145-150.

Bach, E., 1979a,
'Control in Montague grammar',
Linguistic Inquiry 10, 515-531.

Bach, E., 1979b,
'Montague grammar and classical transformational grammar',
in Davis & Mithun 1979, pp. 3-49.

Bach, E., 1980,
'Tenses and aspects as functions on verb-phrases',
in Rohrer 1980, pp. 19-37.

Bartsch, R., 1978,
'Infinitives and the control problem in categorial grammar',
Theoretical Linguistics 5, 217-250.

Bartsch, R., 1979,
'The syntax and semantics of subordinate clause constructions and pronominal reference',
in Heny & Schnelle 1979, pp. 23-59.

Bennett, M., 1976,
'A variation and extension of a Montague fragment of English',
in Partee 1976, pp. 119-163.

Bennett, M., 1977,
'A guide to the logic of tense and aspect in English',
Logique at Analyse 80, 491-517.

Bennett, M., 1978,
'Demonstratives and indexicals in Montague grammar',
Synthese 39, 1-80.

Benthem, J.F.A.K. van, 1977,
'Tense logic and standard logic',
Logique et Analyse 80, 395-437.
Kennis en methode 5, 94-116.

Benthem, J.F.A.K. van, 1981,
'Why is semantics what',
in Groenendijk, Janssen & Stokhof 1981, pp. 24-49.

Bresnan, J.W., 1973,
'Comparative deletion and constraints on transformations',
Linguistic Analysis 1, 25-74.

Chomsky, N., 1965,
'Aspects of the theory of syntax',
The M.I.T. Press, Cambridge (Mass.)

Chomsky, N., 1975,
'Questions of form and interpretation',
Linguistic Analysis 1,
Reprinted in N. Chomsky, 'Essays on form and interpretation',
North Holland, New York, 1977, pp. 25-29.
Cooper, R.H., 1975,
'Montague's semantic theory and transformational syntax',
dissertation, Univ. of Massachusetts, Amherst.
Published by: Xerox University Microfilms.
Cooper, R. & T. Parsons, 1976,
'Montague grammar, generative semantics and interpretative semantics',
in Partee, 1976, pp. 311-362.
Cooper, R.H., 1978,
'A fragment of English with questions and relative clauses',
Unpublished paper, Dept. of Linguistics, Univ. of Wisconsin.
Cooper, R.H., 1979a
'The interpretation of pronouns',
in Heny & Schnelle 1979, pp. 61-92.
Cooper, R., 1979b,
'Variable binding and relative clauses',
in F. Guenthner & J.S. Schmidt (eds), 'Formal semantics and pragmatics for natural languages', Synthese language library 4,
Reidel, Dordrecht, 1979, pp. 131-171.
Curry, H.B., 1961,
'Some logical aspects of grammatical structure',
in 'Structure of language and its mathematical aspects',
Proceedings of Symposia in Applied Mathematics vol. XII,
American Mathematical Society, Rhode Island, pp. 56-68.
Davidson, D. & G. Harman (eds), 1972,
'Semantics of natural language',
Synthese library 40, Reidel, Dordrecht.
Davis, S. & M. Mithun (eds), 1979,
'Linguistics, philosophy and Montague grammar. (Proc. conf. Albany 1977)',
Univ. of Texas Press, Austin, Texas.
Delacruz, E.B., 1976,
'Factives and propositional level constructions in Montague grammar',
in Partee 1976, pp.177-199.
Dik, S.C., 1978,
'Functional grammar',
North Holland Linguistics series 37, North Holland,
Amsterdam.
Dik, S.C., 1980,
'Seventeen Sentences: basic principles and application of functional grammar',
in E.A. Moravcsik & J.R. Wirth (eds), 'Current approaches to syntax', Syntax and semantics 13, Academic Press, 1980, pp. 45-75.
Dowty, D.R, 1976,
'Montague grammar and the lexical decomposition of causative verbs',
in Partee 1976, pp. 201-245.

Dowty, D., 1978,
'Governed transformations as lexical rules in a Montague grammar',
Linguistic Inquiry 9, 393-426.
Dowty, D., 1979a,
'Dative "movement" and Thomason's extensions of Montague grammar',
in Davis & Mithun 1979, pp. 153-222.
Dowty, D.R., 1979b,
'Word meaning and Montague grammar',
Synthese language library 7, Reidel, Dordrecht.
Dowty, D., 1982,
'Grammatical relations and Montague grammar',
in Jacobson & Pullum 1982, pp. 79-130.
Ejerhed, E.I., 1981,
'Tense as a source of intensional ambiguity',
in Heny 1981, pp. 231-252.
Fraassen, B.C. van, 1969,
'Presuppositions, supervaluations and free logic',
in K. Lambert (ed.), 'The logical way of doing things',
Yale Univ. Press, New Haven, 1979, pp. 67-91.
Friedman, J., 1979a,
'An unlabelled bracketing solution to the problem of conjoined phrases in Montague's PTQ',
Journal of Philosophical Logic 8, 151-169.
Friedman, J. & D. Warren, 1979b,
'Erratum',
Linguistics and Philosophy 3, 39.
Friedman, J., & D. Warren, 1980a,
'Notes on an intensional logic for English III. Extensional forms',
Report N-13, Dept. of Computer and Communication Sciences,
The Univ. of Michigan, Ann Arbor (Mich).
Friedman, J. & D. Warren, 1980b,
'Lambda-normal forms in an intensional logic for English'
Studia Logica 39, 311-324.
Gallin, D., 1975,
'Intensional and higher-order modal logic',
Mathematics studies 17, North Holland, Amsterdam.
Gazdar, G., 1980,
'A cross categorial semantics for coordination',
Linguistics and Philosophy 3, 407-409.
Gazdar, G. & I.A. Sag, 1981,
'Passives and reflexives in phrase structure grammar',
in Groenendijk, Janssen & Stokhof, 1981, pp. 131-152.
Gazdar, G., 1982,
'Phrase structure grammar',
in Jakobson & Pullum 1982, pp. 131-186.
Goldblatt, R., 1979,
'Topoi. The categorial analysis of logic',
Studies in Logic and the foundations of mathematics 98,
North Holland, Amsterdam.
Goguen, J., 1978,
'Abstract errors for abstract data types',
in Neuhold 1978, pp. 491-525

Graetzer G., 1968,
'Universal algebra',
The univ. series in higher mathematics, van Nostrand, Princeton.
Second edition published by: Springer, New York, 1979.
Groenendijk, J. & M. Stokhof, 1976,
'Some notes on personal pronouns, reflexives and sloppy identity',
in K. Braunmueller & W. Kuerschner (eds), 'Grammatik. Akten des 10. linguistischen Kolloquiums. Band 2', Max Niemeyer, Tuebingen, 1976, pp. 301-319.
Groenendijk, J. & M, Stokhof, 1979,
'Infinitives and context in Montague grammar',
in Davis & Mithun 1979, pp. 287-310.
Groenendijk J.A.G., T.M.V. Janssen & M.B.J. Stokhof (eds), 1981,
'Formal methods in the study of language. Proceedings of the third Amsterdam colloquium',
MC-Tracts 135 & 136, Mathematical Centre, Amsterdam, 1981.
Groenendijk, J. & M. Stokhof, 1981,
'Semantics of wh-complements',
in Groenendijk, Janssen, Stokhof, 1981, pp.153 -181.
Hausser, R.R., 1976,
'Presuppositions in Montague grammar',
Theoretical linguistics 3, 245-280
Hausser, R.R., 1978,
'Surface compositionality and the semantics of mood',
in Groenendijk & Stokhof 1978, pp. 174-193.
Hausser, R.R., 1979a,
'A constructive approach to intensional contexts. Remarks on the metaphysics of model theory',
Unpublished paper.
Hausser, R.R., 1979b,
'How do pronouns denote?',
in Heny & Schnelle 1979, pp. 93-139.
Hausser, R.R., 1980,
'Surface compositionality and the semantics of mood',
in J.R. Searle, F. Kiefer & M. Bierwisch, 'Speach act theory and pragmatics', Reidel, Dordrecht, 1980, pp. 71-95.
Hausser, R.R., 1984,
'Surface compositional grammar',
'Studies in theoretical linguistics, Fink Verlag, Muenchen.
Henkin, L., 1950,
'Completeness in the theory of types',
Journal of Symbolic Logic 15, 81-91.
Heny, F. & H.S. Schnelle (eds), 1979,
'Selections from the third Groningen round table',
Syntax and Semantics 10, Academic Press, New York.
Heny, F. (ed.), 1981,
'Ambiguities in intensional contexts',
Synthese language library 12, Reidel, Dordrecht.
Hopcroft, J.E. & J.D. Ullman, 1979,
'Introduction to automata theory, languages and computation',
Addison-Wesley, Reading (Mass.).

Hopkins, M., 1972,
'A case for the GOTO',
in 'Proceedings of the ACM (proc. conf. Boston 1972)',
Association for Computing Machinery, New York, 1972, pp. 787-790.
Indurkhya, B., 1981,
'Sentence analysis programs based on Montague grammars',
unpublished paper, Philips international institute
of technological studies, Eindhoven.
Jacobson, P., & G.K. Pullum, 1982,
'The nature of syntactic representation',
Synthese Language Library 15, Reidel, Dordrecht.
Janssen, T.M.V. & P. van Emde Boas, 1977a,
'On the proper treatment of referencing, dereferencing
and assignment',
in A. Salomaa & M. Steinby (eds), 'Automata, languages,
and programming (Proc. 4th. coll. Turku)', Lecture notes in
computer science 52, Springer, Berlin, 1977, pp. 282-300.
Janssen, T., 1978a,
'Compositionality and the form of the rules in Montague
grammar',
in Groenendijk & Stokhof 1978, pp. 101-124.
Janssen, T.M.V., 1978b,
Simulation of a Montague grammar',
Annals of systems research 6, 127-140.
Janssen, T.M.V., 1980a,
'Logical investigations on PTQ arising from programming
requirements',
Synthese 44, 361-390.
Janssen, T.M.V., 1980b,
'On problems concerning the quantification rules in
Montague grammar',
in Rohrer 1980, pp. 113-134.
Janssen, T.M.V., 1981,
'Compositional semantics and relative clause formation
in Montague grammar',
in Groenendijk, Janssen, Stokhof 1981, pp. 237-276.
Janssen, T.M.V., 1981b,
'Montague grammar and functional grammar',
in T. Hoekstra & H v.d. Hulst and M. Moortgat (eds),
'Perspectives on functional grammar', Foris publications,
Dordrecht, 273-297.
also: GLOT 3, 1980, 273-297.
Kamp, H., 1971,
'Formal properties of "Now"',
Theoria 37, 227-273.
Kamp, H., 1975,
'Two theories about adjectives',
in Keenan 1975, pp. 123-155.
Keenan, E. (ed.), 1975,
'Formal semantics of natural language. (Coll. Cambridge 1973)',
Cambridge Univ. Press, 1975.
Keenan, E.L. & L.M. Faltz, 1978,
'Logical types for natural language',
UCLA occasional papers in linguistics 3.

Klein, E., 1979,
On sentences which report beliefs, desires and other
mental attitudes',
unpublished dissertation.
Klein, E., 1981,
'The interpretation of adjectival, nominal and
adverbial constructions',
in Groenendijk, Janssen, Stokhof 1981, pp. 381-398.
Kripke, S., 1972,
'Naming and necessity',
in Davidson & Harman 1972, pp. 253-355.
Kutschera, F. von, 1975,
'Partial interpretations',
in Keenan 1975, pp. 156-174.
Ladusaw, W., 1974,
'Some problems with tense in PTQ',
Texas Linguistics Forum 1, p. 89-102.
Landman, F. & I. Moerdijk, 1981,
'Morphological features and conditions on rules in
Montague grammar',
Amsterdam papers in formal grammar 3, Centrale Interfaculteit
University of Amsterdam.
Landman, F., & I. Moerdijk, 1983a,
'Compositionality and the analysis
Linguistics and Philosophy 6, pp. 89-114.
Landsbergen, J., 1981,
'Adaption of Montague grammar to the requirements of parsing',
in Groenendijk, Janssen, Stokhof 1981, pp. 399-420.
Lewis, D., 1970,
'General semantics',
Synthese 22, 18-67.
Reprinted in Davidson & Harman 1972, pp. 169-248.
Reprinted in Partee 1976, pp. 1-50.
Lewis, D., 1974,
''Tensions',
in M.K. Munitz & P.K. Unger (eds), 'Semantics and
philosophy', New York Univ. Press, New York, 1974.
Link, G., 1979,
'Montague Grammatik. I: Die logische Grundlagen',
Kritische Information 71, Wilhelm Fink, Muenchen.
Loebner, S., 1976,
'Einfuerung in die Montague Grammatik',
Monographien Linguistik und Kommunikationswissenschaft 27,
Scriptor, Kronberg.
Lukasiewics, J., 1920,
'Philosophische Bemerkungen zu mehrwertigen Systemen
des Aussagenkalkuels',
Comptes Rendus des Sceances de Varsovic, Cl. iii, 23, pp. 51-77.
Translated by H. Weber as 'Philosophical remarks on many-valued
systems of propositional logic', in S. McCall (ed.), 'Polish
logic, 1920-1939', Clarendon Press, Oxford, 1967.
Meulen, A. ter, 1980,
'Substances, quantities and individuals. A study in
the formal semantics of mass terms',
distributed by: Indiana Univ. Linguistics Club,
Bloomington (Ind.).

Mikenberg, I., 1977,
'From total to partial algebras',
in A.I. Arruda, N.C.A. da Costa & R, Chuaqui (eds),
Mathematical Logic. Proceedings of the first Brazilian
Conference', M. Dekker, New York, 1977, pp. 203-223.

Montague, R., 1968,
'Pragmatics',
in R. Klibansky (ed.), 'Contemporary philosophy. A survey',
La Nuovo Italia Editrice, Florence, 1968, pp. 102-122.
Reprinted in Thomason 1974, pp. 95-118.

Montague, R., 1970b,
'Universal grammar',
Theoria 36, 373-398.
Reprinted in : Thomason 1974, pp. 222-246.

Montague, R., 1973,
'The proper treatment of quantification in ordinary
English',
in K.J.J. Hintikka, J.M.E. Moravcsik & P. Suppes (eds),
'Approaches to natural language', Synthese Library 49,
Reidel, Dordrecht, 1973, pp. 221-242.
Reprinted in Thomason 1974, pp. 247-270.

Oh, Choon-Kyu, 1977,
'The so-called Bach-Peters paradox explained away',
Theoretical Linguistics 4, 197-207.

Partee, B.H., 1971,
'On the requirement that transformations preserve meaning',
in C. Fillmore & D.T. Langendoen (eds), 'Studies in
linguistic semantics', Holt, Rinehart & Wintson,1971.

Partee, B.H., 1973,
'Some transformational extensions of Montague grammar,'
Journal of Philosophical Logic 2, 509-534.
Reprinted in Partee 1976, pp. 51-76.

Partee, B., 1975,
'Montague grammar and transformational grammar',
Linguistic Inquiry 6, 203-300.

Partee, B.H. (ed.), 1976,
'Montague grammar',
Academic Press, New York.

Partee, B., 1977a,
'John is easy to please',
in A. Zampolli (ed.), 'Linguistic structures processing',
Fundamental studies in computer science 5, North Holland,
Amsterdam, 1977, pp. 281-312.

Partee, B.H., 1977b,
'Possible world semantics and linguistic theory',
The Monist 60, 303-326.

Partee, B.H., 1979a,
'Constraining transformational Montague grammar: A framework
and a fragment',
in Davis & Mithun 1979, pp. 51-102.

Partee, B.H., 1979b,
'Montague grammar and the well-formedness constraint',
in Heny & Schnelle 1979, pp. 275-314.

Partee, B. & E. Bach, 1981,
'Quantification, pronouns and VP-anaphora',
in J.A.G. Groenendijk, T.M.V. Janssen, & M.B.J. Stokhof, 1981,
'Formal methods in the study of language. Proceeding of the third Amsterdam colloquium. Part 2', MC Tract 136, Mathematical Centre, Amsterdam, pp. 445-481.
reprinted in J. Groenendijk, T.M.V. Janssen & M. Stokhof (eds), 'Truth, interpretation and information', Grass 3, Foris, Dordrecht, 1984, 99-130.

Rodman, R., 1976,
'Scope phenomena, "movement transformations", and relative clauses',
in Partee 1976, pp. 165-176.
Gunter Narr, Tuebingen, 1977.

Rohrer, C., 1980,
'Time, tense and quantifiers. (Proc. conf Stuttgart 1979)',
Linguistische Arbeiten 83, Max Niemeyer, Tuebingen.

Saarinen, E., 1978,
'Backwards-looking operators in tense logic and in natural language',
in J. Hintikka, I. Niiniluoto, & E. Saarinen (eds), 'Essays on mathematical and philosophical logic. Proceedings of the 4th. Scandinavian logic symposium', Synthese Library 122, Reidel, Dordrecht, 1978, pp.341-367.

Scott, D. & C. Strachey 1971,
'Towards a mathematical semantics for computer languages',
in J. Fox (ed.), 'Computers and automata (proc. symp. Brooklyn)', Polytechnic Press, Brooklyn (N.Y.), 1971, pp. 16-46.

Sintzoff, M., 1967,
'Existence of a van Wijgaarden syntax for every enumerable set',
Annales de la Societe Scientifique de Bruxelles 81, 115-118.

Thomason, R.H., 1972,
'A semantic theory of sortal incorrectness',
Journal of Philosophical Logic 1, 209-258.

Thomason, R.H. (ed.), 1974,
'Formal philosophy. Selected papers of Richard Montague',
Yale University Press, New Haven.

Thomason, R.H., 1976,
'Some extensions of Montague grammar',
in Partee 1976, pp. 77-117.

Tichy, P., 1971,
'An approach to intensional analysis',
Nous 5, 273-297.

Veltman F., 1981,
'Data semantics',
in Groenendijk, Janssen & Stokhof 1981, pp. 541-565.
Revised reprint in Groenendijk, Janssen & Stokhof, 1984, pp. 43-63.

Visentini et al., 1970,
'Linguaggi nella societa e nella technica',
Edizioni di communita, Milan (distributed by the Olivetti Corporation).

Waldo, J., 1979,
'A PTQ semantics for sortal incorrectness',
in Davis & Mithun 1979, pp. 311-331.
Wijngaarden, A. van, 1970,
'On the boundary between natural and artificial languages',
in Visentini et al., 1970, pp. 165-176.
Wijngaarden, A. van, et al., 1975,
'Revised report on the algorithmic language ALGOL 68',
Acta Informatica 5, 1-236.