Remko J.H. Scha

Logical Foundations for Question Answering

Printed by Eindhoven Druk, Eindhoven
Typesetting: Koningsveld & Zoon BV, Arnhem
Cover design: J. Scha

RIJKSUNIVERSITEIT TE GRONINGEN

LOGICAL FOUNDATIONS FOR QUESTION ANSWERING

PROEFSCHRIFT

ter verkrijging van het doctoraat in de Letteren

aan de Rijksuniversiteit te Groningen

op gezag van de Rector Magnificus Dr. L.J. Engels

in het openbaar te verdedigen

op donderdag 17 februari 1983

des namiddags te 4.00 uur

door Remko Jan Hendrik Scha

geboren te Eindhoven

Promotores:

Prof. Joyce Friedman
en
Prof. Frank Heny

Remko J.H. Scha

LOGICAL FOUNDATIONS FOR QUESTION ANSWERING

M.S. 12.331. Philips Research Laboratories,
Eindhoven, the Netherlands.
1983

Contents.

Chapter III. The PHLIQA1 Question Answering System.

1. Introduction.
2. External Requirements for a Question Answering System.
3. The Top Level Design of PHLIQA1.
4. An English-oriented Level of Meaning Representation.
5. The World Model Language.
6. The Data Base Language.
7. Translations.
8. The Control Structure of the PHLIQA1 Program.
9. The Behaviour of the PHLIQA1 System.

Chapter IV. Data Bases as Value Specifications.

1. Introduction.
2. Value Specifications.
3. Relational Data Bases Viewed as Value Specifications.
4. CODASYL Data Bases Viewed as Value Specifications.
5. Data Bases as Axiom Sets.

Chapter V. Translation Specifications: a Technique for Representing the Conceptual Information of a Question Answering System.

1. Conceptual Information: the Bridge Between Different Levels of Meaning Representation.
2. Conceptual Information in the Form of Translation Rules.
   2.1. Translation Rules.
   2.2. Type Constraints.
3. Translation Between Languages with Different Type Systems.
4. Identification Translations.
   4.1. The Problem of Compound Attributes.
   4.2. More Complex Examples.
   4.3. The Definition of Identification Translations.
5. Simplification Transformations.
6. Extending the Data Base Language.

Chapter VI. Alternative Knowledge Representation Techniques.

1. Introduction.
2. Rule schemes.
   2.1. Introduction.
   2.2. Rule Schemes for First-Order Languages.

Preface.

This thesis has a long history. It began in 1971 at Philips Electrologica in Apeldoorn when Herman Schweigmann suggested to me that I investigate the idea of developing a Natural Language Question Answering System with an ordinary data base as its knowledge base. In response to this suggestion, I developed the concept of a question answering program which would gradually transform the logical representation of a natural language query into a data base query by applying a series of "translation rules". Between 1972 and 1979 a system of this kind, called PHLIQA1, was actually developed at Philips Research Laboratories in Eindhoven.

Wim Bronnenberg, Harry Bunt, Jan Landsbergen, Piet Medema, Wijnand Schoenmakers, Eric van Utteren and myself were the participants in the first phase of this process which ended with the implementation of a Question-Answering System in 1975. Although the system incorporated many new ideas and displayed an interesting structure, its theoretical underpinnings were less than completely satisfactory. Occasionally, this would also show up in incorrect answers or otherwise undesirable responses.

Therefore, in the period from 1976-1978, important aspects of our approach were rethought. The system was redesigned at that time by Wim Bronnenberg, Jan Landsbergen, Wijnand Schoenmakers, Eric van Utteren and myself. In 1979 I wrote an elaborate description of the new program (Bronnenberg et al., 1980) while Bronnenberg, Landsbergen, Schoenmakers and Van Utteren implemented it (within a few months) and debugged it (within a week). The system was successfully demonstrated for an extensive period without displaying any unexpected behavior.

This thesis may be viewed as a theoretical complement to PHLIQA1. It expands on some important ideas underlying this system and compares them to possible alternatives. The issues focussed on here are some of those I feel particularly responsible for in the development of PHLIQA1: the over-all design and the semantics of questions, answers and data.

I wish to thank my former colleagues on the PHLIQA1 Project for everything which they have indirectly contributed to these pages. Particularly, I wish to thank Jan Landsbergen for years of stimulating cooperation, and to acknowledge his share in many aspects of the design of PHLIQA1. His crucial contributions to the development of the method of Translation Specifications, discussed in Chaper V, should be mentioned especially.

In writing this book, I have borrowed freely from some papers which I wrote some time ago, especially Bronnenberg et al. (1980) which forms the basis of Chapter III and Scha (1982) which is incorporated in large part in

Chapter V. The content of Chapter II was developed in a series of talks I have given at the Philosophy Departments of the Universities of Amsterdam, Groningen and Nijmegen.

I would like to thank both of my promoters, Professor Joyce Friedman and Professor Frank Heny, for their thorough and far-reaching comments on both content and form of my earlier drafts. If the present book is readable at all, that is largely due to their efforts and to the extremely active editorial assistance I have received from Dr. Livia Polanyi.

Chapter I. Methodological Preliminaries.

1. **Introduction.**

This book explores some general issues related to the design of question answering systems; the logical representation of the contents of questions and answers, the logical analysis of data bases, and other issues concerning knowledge representation. It explores all these matters from one coherent perspective: the perspective of logical model theory.

Employing Tarski's (1936) idea of representing a state of the world by means of an interpretation of a logical language, precise definitions may be given of the meanings of the various data structures which play a role in question answering programs: representations of the contents of questions and answers, as well as data bases and other forms of "knowledge". This is demonstrated in some detail in the chapters of this book.

Chapter II criticizes existing proposals concerning the model-theoretic semantics of questions and answers, and puts forward an alternative.

The representation of knowledge in a way which dovetails with the logical representation of the contents of questions and answers is addressed later in the book. The goal of providing an interface with an "ordinary data base" will be focussed on particularly. The knowledge representation problem is therefore broken up into two parts. In Chapter IV data bases are construed as well defined objects within the framework of logical model theory, while in Chapter V a proposal is made for how to bridge the semantic gap between the natural language formulation of a query and the notions contained in the data base.

A question answering system which embodies the main ideas developed in this book is described in Chapter III. This system, PHLIQA1, distinguishes itself favourably from other efforts in the field of computational linguistics because of its unusually refined modular structure, featuring explicitly defined interface languages between the modules and precise definitions of their tasks.

In the present introductory chapter, the model-theoretic perspective which constitutes the framework of this book is briefly contrasted with some alternative methodologies which have been brought to bear on similar problems: the *Artificial Intelligence (A.I.)* paradigm (whose adherents might take issue with our very goals of reliability and consistency), the idea of *Procedural Semantics* (which proclaims to be a superior alternative to model-theoretic semantics), and the use of *Predicate Calculus* (which is promoted by a research tradition which is focussed on a particular body of deductive techniques rather than on the use of well-defined meaning representations in general).

2. **Artificial Intelligence.**

The design of natural language question answering programs is often viewed as belonging to the general realm of Artificial Intelligence. The discussion in this book, however, will touch only tangentially on A.I. work on natural language and knowledge representation. This is because much of this work proposes knowledge representation data structures without specifying their semantics with any precision. Much A.I. work shows no interest in developing modular program structures with well-defined interface languages and thus sheds no light on the problem area concentrated upon here.

The difference between the A.I. approach and the one I wish to advocate is not a mere difference of opinion on how to attack a technical problem, but a much more fundamental disagreement about the nature of the enterprise one is engaged in.

A.I. research attempts to design working computer programs for tasks which seem intrinsically to involve the higher intellectual faculties. Natural language processing is often taken to be a paradigm example of this kind of task. In A.I., the goal of designing computer programs capable of performing tasks of this sort efficiently and correctly is taken to be somehow equivalent to the goal of modelling central aspects of human cognition. In my view, however, these enterprises are entirely different and there can be no virtue in confusing them. Let me expand on this somewhat.

A fundamental difference between the two activities is that they have different goals which overlap only slightly. Humans have many characteristics, for example, which one would just as soon do without in a computer program. We are often sloppy and unreliable, frequently forgetful and uncooperative. Therefore, if one could make a question answering system equalling human question answering performance, one should try to make a system which exceeds it. If the goals of question answering, automatic translation, expert systems and the like are contrued as "cognitive modelling", then one is, in fact, undercutting the potential which electronic computers may have for such tasks.

Whether implemented computer programs can be expected to be structurally similar in any way to human mental processes is also open to serious doubt. The basic underlying hardware processes are so different in both cases that it is by no means self-evident that corresponding structures can be used to implement corresponding tasks.

Those working within hardcore A.I. tend to avoid facing the choice between doing A.I. as applied computer science or as theoretical psychology. By failing to deal clearly with the goals and purposes of the A.I. activity, practitioners need demonstrate neither correctness and efficiency in their

programs nor psychological validity. Comparisons with the human mind can be freely invoked to justify a lack of interest in the correctness, consistency and modularity of a program while the extent to which the program constitutes a model for human mental processing structures is left unspecified.[1] In light of this, it is best to view A.I. not as a scientific or technological discipline in the way in which these notions are normally considered but to see it rather as the experimental branch of computer science where new programming concepts are tried out without much concern for their theoretical underpinnings and unfettered by the constraints imposed by well-defined requirements of any sort.

This explains why one would not expect to deal in depth with A.I. work in natural language understanding in a discussion which is involved in evaluating different, well-described means for achieving the same (or similar) well-defined goals. This is why, in the rest of this book, references to A.I. work are made only where they are specifically relevant to the matter under discussion.

3. **Model-Theoretic Semantics vs. Procedural Semantics.**

An important goal of the discussion in this book is to give definitions of notions such as the content of a question, the content of an answer, and knowledge about a subject domain in such a way that precise requirements for the correct operation of a question-answering program may be formulated. Elsewhere in the discussion, it will be shown that the notions of *logical model-theory* can be used very well for this purpose. In the present section, the virtues of model theory will be compared briefly with the properties of *procedural semantics*, a rather different formulation of semantics which developed out of Computational Linguistics and Artificial Intelligence.[2]

Model-theoretic semantics can be used as a tool to define the meaning of data structures independently of the programs which access them and to define the task of program modules independent of their implementation. A model-theoretic description of the content of a question specifies what answers would be correct in terms of a precisely defined notion of *state of the*

[1] A rather extreme position on this matter, which I would not want to ascribe to the A.I. Community as a whole, dismisses altogether the notion of a model as having an explicitly specified structural correspondence to the phenomenon it is supposed to model. Instead, it is simply stipulated that object A is a model for phenomenon B if people can be led to mistake A for B. This decision to declare mimesis to be the highest goal of the cognitive sciences is often credited to a philosophical essay by Turing (1950) which proposes a kind of "mimetic reductionism" of mentalistic notions. The work done by Colby constitutes the most explicit exemplification of this point of view. See, for instance, Colby et al. (1972) and Colby (1975).

[2] See Woods (1968, 1981); Woods et al. (1972); Winograd (1972).

world. How the answers will be computed is left open by this specification. Similarly, a model-theoretic description of the meaning of a piece of data shows how it constrains the states of the world which are considered possible, without specifying how this information is to be brought to bear on the queries. Thus, model-theoretic semantics may be used to introduce a considerable measure of precision into the discourse surrounding the structure and operation of a question-answering system without complicating matters with purely algorithmic considerations.

In procedural semantics, on the other hand, the meaning of a question is said to be a *procedure* for computing the answer.

One problem arises because this procedure is assumed to be formulated in terms of certain primitive procedures which need not be further analyzed. But for the "primitive procedures" which yield factual knowledge concerning contingent states of affairs in the world, this view is not justified. Their values may not be known at the moment of the design of the system; it must be possible to change them if the state of affairs in the world changes. Therefore, we do not want to view these procedures as unanalyzed entities; they must be viewed as either containing or accessing data structures of some sort, and the meanings of these data structures must be described precisely.

Another problem with procedural semantics as it has been defined until very recently,[3] is that it is too specific. Because the meaning of an expression encompasses its *evaluation method* there is no room for logical equivalence transformations such as those which turn an expression into an equivalent but more efficiently evaluable one. The general criticism which should thus be made about this framework, from the point of view of model-theoretic semantics, is that it is conceptually too poor.

4. **Limitations of Predicate Calculus.**

When methods from formal logic are used in knowledge representation and natural language analysis, the formalism used is often the first-order predicate calculus. In fact, the notions of "logic" and "first-order predicate calculus" are all but interchangeable in the work of certain authors. (See, for example, the papers collected in Gallaire and Minker, 1978.)

Most syntactically and semantically well-defined languages used in computational systems for semantic representation derive more or less directly from first-order predicate calculus. (Woods, 1968; Green, 1969; Petrick, 1973). Proposals about semantic nets often pertain to pictorial representations of first-order predicate calculus expressions as well (Schubert, 1976).

[3] Woods, the foremost exponent of procedural semantics, has kept redefining it, to the extent that it is now indistinguishable from model-theoretic semantics. Sentences are now analyzed as "abstract partial procedures". (Woods, 1981)

It must be emphasized, therefore, that the approach taken in this book does not embody any preconceptions about the syntactic form of the logical representation languages used. What has been of overriding concern, however, is that the expressions in the languages used be precisely defined and that their semantics (in the model-theoretic sense) be precisely defined as well. The first-order predicate calculus was rejected as a possible logical formalism for the general task of natural language modelling, because the semantic phenomena involved can not be accommodated within the limitations of this framework. For example, "collective quantification" requires functions or predicates on sets of individuals while "cumulative quantification" requires variables ranging over Cartesian products (i.e. over sets of n-tuples) and selection-operations on n-tuples. (Scha, 1981) Comparatives result as well in quantificational structures not expressible in the minimal extensions of first order logic which are normally made.[4] "Bags" are necessary for dealing with the "exclusive-or" operator (Borowski, 1976) and with certain noun phrase denotations.[5] In addition, lambda-abstraction is necessary for using the technique of "translation specification", a most important possibility exploited here for knowledge representation.[6]

Therefore, a richer formal language than first-order predicate calculus was necessary to do the tasks we had in mind adequately: the languages introduced below as EFL (English-oriented Formal Language), DBL (Data Base Language) and WML (World Model Language) have the power needed to accomplish them. Appendix A specifies the syntax and semantics of these languages.

5. **Conclusion.**

To be able to design a reliably operating question-answering system with a modular structure, it is necessary to define notions of questions, answers and knowledge which are suited for a computationally effective treatment. These notions must be based on an explicit semantics, moreover, which makes it possible to formulate explicit correctness requirements for the algorithms employed in the system. In sections 2 and 3, above, the A.I. and procedural semantics approaches to question-answering system development were rejected because they are unequipped to deal with constraints of this sort. In section 4 the first-order predicate calculus was also rejected as a suitable framework because it lacks the richness and power needed to deal with linguistic complexities adequately. Let us now go on to tackle the main

[4] *Akzo bought more computers than Philips sold.*

[5] *What is the sum of the prices of Akzo's computers?*

[6] See Chapter V.

subject areas we shall be dealing with: the semantic and computational aspects of the representation of questions, answers and knowledge. Because it is basic to the entire issue we address here, we will begin with the semantic analysis of questions and answers.

Chapter II. Questions and Answers.

1. **Introduction.**

The English language possesses specific sentence categories whose primary function is to express a request for information.[1] Sentences belonging to these categories are called "interrogative sentences" or "questions". According to the rules of English discourse, a question asked by one discourse participant to another creates a positive obligation on the part of the addressee either to provide the requested information (i.e. give an "answer"), or to explain why he doesn't (give a "reply" in a more general sense). This chapter contains a discussion of formal methods for representing the contents of questions and the contents of answers, which make it possible to account for semantic relations between questions and answers - most importantly, for the *correctness* of a given answer with content A in response to a question with content Q.

We should like to characterize the content of a question ("what it is that a question asks") and the content of an answer ("what it is that an answer describes") in such a way that a precise account can be given of how the correctness of an answer depends on the state of the world. Whereas the contents of "isolated assertions" have received much attention in studies concerning philosophical logic and formal semantics (where they are analysed as "propositions", i.e. as functions from interpretations or possible worlds to truthvalues), there is not yet much agreement about the proper semantic analysis of questions and answers. It should be pointed out here, that answers cannot be equated with "isolated assertions". Answers are not "stand-alone" utterances: they depend on context for proper interpretation.

The syntactic forms of answers come in such a wide variety of forms, that one may doubt whether they should all be treated as expressing the same kind of semantic object. The examples (1)-(15) give an impression of the variety of possible answers of the same question.

Did John go to the movies yesterday? (1)

may be answered by

Yes. (2)

Yes, he did. (3)

That's what he did. (4)

He went to the movies yesterday. (5)

He finally went. (6)

[1] Secondary functions of questions have been widely recognized. Discussions of such secondary uses of questions (e.g. Churchill, 1978; Goody, 1978) indicate that they may always be analysed as "parasitic" on the primary use of requesting information.

No, Susan's out of town. (7)
Susan's out of town. (8)

The range of possible answers to a wh-question is just as wide.

Which girls did you like at Bill's party? (9)

may be answered by

Jane and Mary (10)
The girls we almost ran into in the hallway (11)
None. (12)
I liked Jane and Mary. (13)
Jane and Mary are the ones I liked. (14)
Well, the only friend of Bill's that I can barely stand at all is Susan, and she's out of town you know. (15)

This variety of examples also suggests that it would be attractive to treat certain forms of answers as semantically "basic", and to explain other forms as somehow "derived"; but what kinds of answers to pick out for the privileged status of "semantically basic" is still a matter of debate, as we shall see.

The next section will sketch and criticize some proposals which analyse questions and answers in terms of propositions (Hamblin (1973), Karttunen (1977), Groenendijk and Stokhof (1981)). Proposals of this sort assume that answers may be analysed independently of the question, as expressing propositions. This assumption is challenged in section 3. Section 4 discusses a proposal by Hausser (1980) which, for wh-questions, focusses on answers which are noun phrases rather than full sentences. Undesirable features of his treatment are pointed out.

Section 5 presents my own proposal, which also focusses on "short answers". This proposal, which provides the theoretical background for the treatment of questions and answers in the question-answering system PHLIQA1 (Medema et al., 1975; Bronnenberg et al., 1978), is closer in spirit to Whately (1826) and Tichý (1978) than to Hausser.

One of the points which will emerge from the discussion in this chapter, is that semantic correctness is only one of the conditions that an adequate answer must fulfill. A computer program for question answering must also embody pragmatic strategies which determine what kind of a correct answer it wants to give. This issue is discussed in the final section of the present chapter.

2. Characterizing the Content of a Question in Terms of its Propositional Answers.

2.1. Hamblin: Questions as Sets of Possible Answers.

Hamblin's (1973) extension of Montague's "English as a Formal Language" (1970) treats questions and answers. Answers are implicitly equated with isolated assertions, and analysed as propositions.[2] Questions are analysed as sets of propositions: the propositions expressed by possible answers to the question. A correct answer can then simply be defined as a proposition which is true and is included in the proposition set expressed by the question.

The possible propositional answers to a yes/no question in Hamblin's analysis are, plausibly enough, the assertion corresponding to the question, and the negation of that assertion. A yes/no question is therefore analysed as a set which contains a proposition and its negation. For instance,

Does John walk? (1)

would be analysed as

$\{\hat{}\text{WALK}(\text{JOHN}),\ \hat{}\neg\ \text{WALK}(\text{JOHN})\}$ (2)

Besides yes/no questions, Hamblin treats singular wh-questions, like

What dog walks? (3)

which are interpreted as "mention-one" questions; e.g., (3) is read as a request to assert about some entity that it is a walking dog. The possible answers to (3) that Hamblin considers are sentences like

Fido is a dog and he walks. (4)
Fido is a dog which walks. (5)
Fido is a walking dog. (6)

which express propositions which can be represented by formulas of the form

$\text{WALK}(a)\ \&\ \text{DOG}(a)$ (7)

[2] Strictly speaking, Hamblin analyses assertions as sets of propositions. But unlike most questions, they are singleton sets.

where *a* is a logical proper name. Question (3) is therefore analysed as the set of propositions

$$\{P \mid \exists\ x\text{: } P = \hat{}\,(\textsc{walk}(x) \,\&\, \textsc{dog}(x))\} \tag{8}$$

Hamblin focusses on "mention-one" readings of wh-questions, but this is not because of any essential properties of his approach. For example,

Which dogs walk? (9)

may be read as a "mention-all' question, requesting its addressee to assert about some set that its elements are all the dogs that walk. In Hamblin's spirit, this question (i.e., its possible answers) would be represented by the formula

$$\{P \mid \exists\, X\text{: } P = \hat{}\,(X = \{x \mid \textsc{dog}(x) \,\&\, \textsc{walk}(x)\})\} \tag{10}$$

Hamblin's representation of the content of a question means that, at the logical level, answering a question with content Q would consist in finding an "explicit" formula (i.e., one which does not quantify over propositions) which denotes a proposition $A \in Q$ such that $\check{}A$. For yes/no questions this would amount to finding out which one of the two possible answers denotes TRUE. For wh-questions, where the set of possible answers is defined by a formula of the form

$$\{P \mid \exists\ x\text{: } P = \hat{}\,\mathrm{R}(x)\}, \tag{11}$$

finding the logical representation of an answer would consist in finding a logical proper name *i* such that R*(i)*.

2.2. **Karttunen: Questions as Properties of True Answers.**

Karttunen's (1977) proposal for dealing with the semantics of questions derives from Hamblin's but differs from it on some points. He describes an extension of Montague's "Proper Treatment of Quantification in Ordinary English" (Montague, 1973) which accommodates indirect questions (embedded whether- and which-clauses). The analyses of whether- and which-clauses are intended to carry over directly to yes/no questions and which-questions, respectively.[3)]

[3)] Karttunen proposes to do this by equating questions with assertions of the form "I ask you to tell me..." I do not see how this could have any advantages above assuming the usual speech act analysis which distinguishes different illocutionary forces such as questioning, asserting, etc. Nothing hinges on this aspect of Karttunen's proposal, however. What I call "the content of the question", i.e. the semantic object of the "question-operator", would correspond to the direct object of the relation representing the verb "tell" in Karttunen's treatment.

Karttunen views answers as propositions. These answer-propositions are not to be equated with isolated assertions, however, but must be "processed" in the "context" of the question they answer. A question is analysed as the property of those propositions which correspond to true answers. A (full sentence) answer with content A in response to a question with content Q can then be defined to be correct simply iff $\check{}Q(A)$.

Possible answers to yes/no questions are defined just as in Hamblin's treatment. Wh-questions are again viewed as mention-one questions, but they are treated in a slightly different way. For instance, the possible answers to

Which girl sleeps? (1)

which are considered, are not the answers of the form

Mary is a girl who sleeps. (2)

but the answers of the form

Mary sleeps. (3)

The possible answers in Karttunen's treatment are the answers expressing propositions which can be represented by formulas of the form

Sleep (i) (4)

such that i is a logical proper name and GIRL (i) is true. Question (1) is therefore analysed as the propositional concept

$$\hat{}\{\lambda P: \check{}P \ \& \exists\, x: \ \text{GIRL}(x) \,\&\, P = \hat{}\,\text{SLEEP}(x)\} \tag{5}$$

Analysing the content of a question as the property of its true answers, as Karttunen does, rather than the set of its possible answers, as Hamblin proposed, does not lead to essentially different consequences. The same class of answers is accounted for, in an equally simple way. Because of this, one might prefer the Hamblin treatment, since the content of a question is a "simpler" object in this treatment; Karttunen assigns a higher level of intensionality to the content of a question than Hamblin does.

2.3. **Answer-Propositions in Context.**

The present subsection discusses a departure from Hamblin's ideas, made by Karttunen's treatment, concerning the decision as to what to take as a paradigmatic propositional answer to a mention-one question. This decision has immediate consequences for the role which the meaning of the "wh-nounphrase" plays in the content of a question. Consider question (1) again,

Which girl sleeps? (1)

with its two answers (2) and (3).

Mary is a girl who sleeps. (2)

Mary sleeps. (3)

If (1) is answered by (3), this utterance of (3) "implicitly states" (as Hamblin puts it) that Mary is a girl. The propositions which Hamblin counts as answers are therefore analyses of sentences like (2) rather than (3), which means that answers like (3) have to be treated as somehow elliptical.

Karttunen's answers have the form (3) rather than (2); this amendment suggests a different kind of account of the situation, which has attractive features. There is some plausibility in the idea that an answer should not be treated as an isolated assertion, but should be processed in the context created by the question instead. In this case, what an answer communicates does not necessarily coincide with the assertion of the truth of the expressed proposition; instead, it communicates the (possibly more specific) fact that this proposition constitutes a true answer to the preceding question. What an answer with content A communicates in the context of a question with content Q is the information that $\check{}A \;\&\; (A \in Q)$. For instance, in the context of a question with content

$$\{P \mid \exists\, x\text{: } \textsc{girl}(x) \;\;\&\;\; P = \hat{}\,\textsc{sleep}(x)\} \tag{4}$$

(i.e., assuming the Karttunen amendment of Hamblin's proposal as far as the placement of the predicate GIRL is concerned), an answer with content

$$\hat{}\,\textsc{sleep}\,(\textsc{mary}) \tag{5}$$

communicates the information that

$$\textsc{sleep}\,(\textsc{mary}) \;\&\; (\hat{}\,\textsc{sleep}\,(\textsc{mary}) \in \{P \mid \exists\, x\text{: } \textsc{girl}(x) \;\;\&\;\; P = \hat{}\,\textsc{sleep}(x)\} \tag{6}$$

which is logically equivalent to

$$\textsc{sleep}\,(\textsc{mary}) \;\&\; \textsc{girl}\,(\textsc{mary}) \tag{7}$$

Thus what is "implicitly stated" by (3) is exactly the difference between what it communicates as an isolated assertion (i.e., the proposition it expresses) and what it communicates in the context of the question.

A problem with Karttunen's variant arises, however, when readings of wh-questions which require exhaustive answers are considered. If

Which girls sleep? (8)

is read as requesting the assertion of a proposition which indicates the extension of the set of sleeping girls, the set of its possible answers may be represented as

$$\{P \mid \exists X: P = \hat{}(\{x \mid \text{GIRL}(x) \,\&\, \text{SLEEP}(x)\} = X)\} \tag{9}$$

There seems to be no way of avoiding that the predicate GIRL is involved in the answer propositions, if these are required to be exhaustive.

2.4. **Groenendijk and Stokhof: Exhaustiveness.**

Groenendijk and Stokhof (1981, 1982) propose an alternative extension of Montague's (1973) "Proper Treatment of Quantification in Ordinary English", in which they try to accommodate embedded wh-clauses. They mention the possibility that their analyses might carry over to direct question sentences, although they express strong reservations as to whether this is in fact the case. Groenendijk and Stokhof's proposal differs from Karttunen's PTQ-extension in two important ways. One is, that they treat "mention-all" readings of wh-clauses, rather than "mention-one" readings. The second difference is, that they describe the propositional concepts which constitute the contents of questions in a different way than Karttunen.

To handle certain complications concerning the interaction between wh-noun-phrases and "ordinary" quantifiers, Groenendijk and Stokhof introduce nested abstractions over possible worlds – something which cannot be expressed in Montague's intensional logic IL. Groenendijk and Stokhof therefore adopt TY2 (Gallin, 1975) – a variant of IL which has variables ranging over possible worlds, and can therefore express intension- and extension-operators by lambda-abstraction and function-application respectively. Groenendijk and Stokhof's treatment, applied to direct questions, would analyse the content of a wh-question as a property of propositions (a "propositional concept") – the property of being a true answer to the question. For instance:

Who walks? (1)

expresses the propositional concept

$$(\lambda w: (\lambda i: \mathbf{ext}_i(\text{WALK}) = \mathbf{ext}_w(\text{WALK}))) \tag{2}$$

(Notation: w and i range over possible worlds; $\mathbf{ext}_j(F)$ is a mnemonic notation for $F(j)$, that I use when j is of the type "possible world", standing for "the extension of F in j".)

Assuming this treatment, an answer with content A given in reply to a question with content Q is correct iff $\check{}Q = A$.

The possibility of accounting for the mention-all reading of wh-questions is clearly important for a computer system providing automatic question-

answering services. In such situations, mention-all readings often are more plausible than mention-one readings. I also find some plausibility in Groenendijk and Stokhof's suggestion that the mention-all reading is needed in the semantics of wh-clauses embedded under "know". Nevertheless, wh-questions can under certain circumstances be adequately answered by giving partial information. In section 5.4 I shall come back to this and present a treatment which covers both cases.

2.5. **Problems with Rigid Designators.**

In the present section I shall take a closer look at the kinds of answers which are accounted for by the treatments of questions and answers in the previous subsections. Consider, for instance, the content of the question

Who walks? (1)

which Hamblin's treatment analyses as

$(\lambda P: \exists x: P = \hat{}\,\text{WALK}(x))$ (2)

According to this treatment, a valid answer to this question would be any sentence asserting a true proposition which can be expressed by a formula of the form

$\hat{}\,\text{WALK}(b)$ (3)

where b is a rigid designator. As examples of such assertions, sentences like

Mary walks. (4)

are usually cited.

Answers of this kind are problematic at two levels. It is problematic whether they ought to be expressible in the logical language, and it is problematic whether they are expressible in natural language. I shall now discuss both these problems.

There are intrinsic problems with the assumption that an intensional logical language contains a logical proper name for every individual.[4] It implies that the number of individuals in the domain is denumerable, and the same for all possible worlds. This severely constrains the descriptive power of the language. (See Potts, 1976.)

[4] In an extensional language this is less problematic. See footnote in Chapter IV, Section 2.

One curious consequence of the assumption of a fixed set of rigid designators is demonstrated by Groenendijk and Stokhof's system, in which

Who walks? (1)

would be analysed as

$$(\lambda w\text{: } (\lambda i\text{: } \mathbf{ext}_i\,(\textsc{walk}) = \mathbf{ext}_w\,(\textsc{walk}))) \tag{5}$$

thus requiring a correct answer to specify the complete extension of the predicate WALK, i.e. to specify for every individual in the domain whether it walks or doesn't. As Karttunen (1977) noticed in rejecting a different treatment with this property, this seems to be asking too much. The question (1) asks for a specification of those who *do* walk, not of those who do not. A correct answer to (1) is not expected to give the same information as a correct answer to

Who doesn't walk? (6)

It might seem easy to devise a variant of Groenendijk and Stokhof's treatment which is less overdemanding: analyse (1) not as (5), but as

$$(\lambda w\text{: } (\lambda i\text{: } \mathbf{ext}_i\,(\{x \mid \textsc{walk}(x)\}) = \mathbf{ext}_w\,(\{x \mid \textsc{walk}(x)\}))) \tag{7}$$

However, in a language with rigid designators for all individuals, (7) is equivalent to (5).

Another problem with the propositions in terms of logical proper names is that they usually cannot be expressed in natural language. The English words for the integers may be viewed as corresponding to rigid designators. The case of proper names for people is already more complicated; and for most things, natural languages provide no proper names, nor other means of rigidly referring to them.

Besides answers in terms of proper names, other kinds of answers must therefore be accounted for: answers which identify an individual in terms of contingent properties, as in

The hungriest dog in the neighbourhood walks. (8)

and answers which do not uniquely identify any individuals at all, as in

Some dog walks. (9)

It should also be noted that answers of the form (8) or (9) may sometimes be more interesting for the questioner than an answer in terms of proper names. This matter is taken up again in section 6.

3. **Against the Primacy of Full-Sentence Answers.**

The treatment of questions and answers that will be developed in section 5 of this chapter comes from a rather different perspective than the treatments discussed so far. One important difference concerns the decision as to what kinds of answers to treat as "basic". In the treatment presented here, answers which have the form of a noun phrase are viewed as the basic ones, whereas the proposals in the Hamblin tradition assume that an answer has the form of a complete sentence expressing a proposition. The present section compares the merits of these alternative views.

The decision to treat sentences rather than noun phrases as "basic answers" is defended explicitly by Belnap and Steel (1976). As an example they discuss the question

> *What is the freezing point of water, in degrees Fahrenheit, under standard conditions?* (1)

They write:

> *Suppose the respondent replies to (1), not with the full sentence "The freezing point of water under standard conditions is 32°F", but merely with the noun "32". Obviously, its status as an answer, indeed, its very meaning, depends upon the context of its utterance. So, since we are now preparing the way for a formal analysis in which we shall not want any assertoric meanings to be dependent on context, we shall not count "32" as a direct answer to (1). Rather, we shall treat it as merely an abbreviated way of saying "The freezing point of water under standard conditions is 32°F", and we shall call it (after Hamblin 1958) a coded answer. Coded answers, including gestures and nods as well as words, are, because of their efficiency, of enormous importance in communication, but they must always be code for complete and unabbreviated sentences.* (p. 14)

Belnap and Steel express in an explicit way an idea which is implicit in many other treatments of questions: the assumption that a declarative sentence which is uttered in answer to a question may be viewed as independently expressing a proposition, in the same way as the contextless isolated assertions which have so far been the main topic of study of philosophical logic and formal semantics. This assumption, however, is false. One phenomenon demonstrating this relates to an often observed ambiguity which is exhibited by declarative sentences when they are taken out of context. For instance,

John went out with Mary yesterday. (2)

may express various things, such as:

What happened yesterday is that John went out with Mary. (3)
Mary is the one that John went out with yesterday. (4)
John is the one who went out with Mary yesterday. (5)
Yesterday is when John went out with Mary. (6)
It is the case that John went out with Mary yesterday. (7)

As Whately noted as early as 1826,[5] precisely this ambiguity disappears in the context of a question – the topics in the answers depend on the "questioned" elements in the questions. For instance, the sentences (3) - (7) above would be appropriate paraphrases of answer (2) in the context of the following questions:

What happened yesterday? (8) for (3)
Who did John go out with yesterday? (9) for (4)
Who went out with Mary yesterday? (10) for (5)
When did John go out with Mary? (11) for (6)
Did John go out with Mary? (12) for (7)

From this we may infer that any answer to a question must be interpreted in the context of the question it answers. This holds for full-sentence answers just as well as for "minimal answers" in the form of noun phrases. There is thus no reason, from this point of view, to give one or the other a privileged status.

Before presenting our own treatment of questions and answers, which takes minimal answers as "basic", in section 5, another treatment which shares this feature is discussed in section 4. The comparison between the full-sentence answer and the minimal answer approaches will be taken up again in section 5.5.

4. **Hausser: Answers as Noun Phrases.**

Hausser (1980) presents a PTQ extension dealing with questions and answers which, unlike the Montague-style treatments discussed before, focusses on "minimal" answers. Wh-questions are assumed to be answered by noun phrases.

Hausser's treatment of wh-questions is rather limited. It does not treat questions involving noun phrases with "which", for example, but only wh-

[5] See Prior and Prior (1955).

questions of the form "who + verb phrase" or "what + verb phrase" - i.e., the case where the range of the "querification" does not have to be explicitly described in the expression representing the question, but where this range may be assumed to be a semantic type of the logical language.
Hausser analyses the content of a question as a property of NP-denotations. For instance

Who walks? (1)

is analysed as

$(\lambda P\colon P(\text{WALK}))$. (2)

The content of a minimal answer is analysed as an NP-denotation, where an NP-denotation is construed like in Montague (1973), as a function from one-place predicates to truthvalues.
For instance, the minimal answer

A boy. (3)

is analysed as

$(\lambda F\colon \exists\, x \in \text{BOYS}\colon F(x))$. (4)

An answer with content A in response to a question with content Q is correct iff $Q(A)$. This correctness criterion is too tolerant, since it accepts as a correct answer any noun phrase which, combined with the question, yields a true sentence. For instance, if those who walk are the boys John, Peter and Harry, all of the following answers to question (1) are treated as correct:

John, Peter and Harry. (5)
John (6)
No girls (7)

Let us now see how Hausser's approach works if it is applied to questions involving "which-nounphrases". Mention-one readings of such questions can be brought under the scope of Hausser's treatment without much difficulty. For instance,

Which boys walk? (8)

when viewed as a paraphrase of

Who is one of the boys that walk? (9)

may be represented as

$$(\lambda P\text{: } P((\lambda x\text{: } x \in \{y \in \text{BOYS} \mid \text{WALK}(y)\}))) \quad (10)$$

There is a problem, however, with mention-all readings of which-questions. Consider, for instance, the mention-all reading of (8), which may be paraphrased as

Who are the boys that walk? (11)

A representation like

$$(\lambda P\text{: } P((\lambda X\text{: } X = \{y \in \text{BOYS} \mid \text{WALK}(y)\}))) \quad (12)$$

would lead to the acceptance of answers which are, to say the least, quite misleading. For instance, in any state of affairs where at least one happy boy walks, the answer

No sad boys (13)

represented, for instance, as

$$(\lambda F\text{: } \neg \exists y \in \boldsymbol{P^*}(\{z \in \text{BOYS} \mid \text{SAD}(z)\})\text{: } F(y))\} \quad (14)$$

would count as correct.[6)]

An advantage of Hausser's "tolerance", however, is that it can account effortlessly for certain question/answer pairs which are problematic in treatments which implement more specific requirements concerning the question/answer relationship.
Consider, for instance, the question

Who does every man love? (15)

with the answer

A woman (16)

in the reading where a different man may love a different woman. Within

[6)] Notation: $\boldsymbol{P^*}(X) =_{\text{def}} \{Y \mid Y \quad X \,\&\, Y \neq \emptyset\}$

Hausser's approach, there is no problem in giving one of the readings of (15) the representation

$$(\lambda P\text{: } \forall x \in \text{MEN: } P\ (\lambda y\text{: LOVE}(x,y))) \tag{17}$$

while the answer (16) would be represented as

$$(\lambda F\text{: } \exists z \in \text{WOMEN: } F(z)) \tag{18}$$

which, in combination with the representation of the question, would yield the proposition

$$\forall x \in \text{MEN: } \exists z \in \text{WOMEN: LOVE}(x,z) \tag{19}$$

5. The PHLIQA1 Treatment: Questions and Answers as Describing Sets of Individuals.

5.1. Introduction.

The previous sections reviewed some recent proposals which try to give an explicit semantic analysis of the contents of questions and answers. Section 2 discussed treatments which analyse the contents of questions in terms of their propositional answers, and indicated serious problems with these treatments (section 2.5). Section 3 showed that there is in fact no basis for a privileged status of propositional answers. Section 4 therefore turned to Hausser's proposal, which treats short answers as semantically basic, but found his treatment to the question-to-answer relationship too "tolerant" for our purposes.

The present section presents an extended version of the theory underlying the treatment of questions by the question answering system PHLIQA1 (Medema et al., 1975; Bronnenberg et al., 1980). This theory, like Hausser's, treats short answers as basic; in its semantic properties it differs considerably from both Hausser's treatment and the propositional treatments. Subsection 5.2 of the present section discusses the analysis of questions. Subsection 5.3 discusses the analysis of answers and the question-to-answer relationship for a limited version of the theory, which only deals with definite answers. Section 5.4 discusses a revised version, which deals with indefinite answers as well. Section 5.5 discusses full-sentence answers from the perspective of our theory.

5.2. **Questions.**

The perspective on questions which is embodied in the theory presented here may be formulated as follows.[7] A yes/no question presents a proposition (a function from states of affairs to truthvalues), and requests its addressee to indicate which value this function has for the actually obtaining state of affairs. Similarly, a wh-question presents a function from states of affairs to sets of individuals, and requests the addressee to indicate what value this function has for the actually obtaining state.

Thus, every question is viewed as describing an object: a set of individuals in the case of a wh-question, a truthvalue in the case of a yes/no question. The answer to a question must then give a different identification of that same object. The answer to a yes/no question must specify whether the truthvalue is TRUE ("Yes") or FALSE ("No"), while the answer to a wh-question may name all the designated individuals. It is not difficult to decide how a theory embodying this perspective will represent the content of a question: this content is represented by an expression which denotes the object which the question describes. For instance,

Which boys walk? (1)

is represented as

$$\{x \in \text{BOYS} \mid \text{WALK}(x)\} \tag{2}$$

while

Does John walk? (3)

is represented as

$$\text{WALK}(\text{JOHN}). \tag{4}$$

Multiple wh-questions

Multiple wh-questions can be treated by extending the treatment of simple wh-questions in a straightforward way. Such questions describe a set of n-tuples rather than a set of individuals.

[7] The view put forward here has a long history. Whately (1826) expressed ideas which clearly went in this direction (see Prior and Prior, 1955). He emphasizes particularly the uniformity between wh-questions and yes/no questions which is, at the semantic level, achieved by this perspective. Tichý (1978a) formulates it in terms that I find sympathetic as well. His distinction between empirical and mathematical questions, however, seems to be difficult to maintain as linguistically valid. (See section 7.1, note 1, for further discussion of this point.)

Which boys love which girls? (5)

is thus represented as

$$\{u \in \text{BOYS} \times \text{GIRLS} \mid \text{LOVE}(u)\} \quad (6)$$

(Notation: the variable u ranges over ordered pairs; the operator $\times$ forms the Cartesian product of two sets; n-place relations are rendered as predicates on n-tuples, so LOVE is a predicate on pairs. Thus, (6) denotes the set of ordered pairs containing a boy and a girl such that the boy loves the girl.)[8)]

5.3. **Answers.**

Among the wide range of possible answers to questions, the treatment presented here focusses on the shortest possible forms: the so-called "minimal answers". In Section 5.5 below, I shall indicate how more elaborate formulations may be accounted for in terms of this treatment of minimal answers and present further arguments showing that this perspective compares favorably with the more usual procedure of treating the shorter forms as elliptical forms of full sentences.

A minimal answer to a yes/no question is either "yes" or "no" while a noun phrase identifying or describing the set of individuals belonging to the set described by the question is a minimal answer to a wh-question. For instance:

Which girls at Susan's party did you like? (1)

has all of the following responses among its possible minimal answers:

Jane and Mary. (2)

The girls that we almost ran into in the hallway. (3)

None. (4)

Some of the girls that Bill brought. (5)

[8)] Quantification over Cartesian products is independently motivated. It is needed to represent readings of sentences which exhibit the phenomenon of "cumulative quantification". An example of this may be observed in the sentence

600 Dutch firms have 5000 American computers.

when we read it as being equivalent to

The number of Dutch firms which have American computers is 600, and the number of American computers possessed by Dutch firms is 5000.

Elsewhere I have shown how such readings may be systematically generated by allowing noun phrases to combine into quantifiers which have Cartesian products (e.g., in this case, the product of the set of Dutch firms and the set of American computers) as their range (see Scha, 1981). Therefore, this manner of rendering multiple wh-questions is more attractive and less ad hoc than it may first appear.

Among these answers, definite and indefinite answers should be distinguished. A definite answer has a form which shows that if it is felicitously used at all, it uniquely defines one set of individuals (whether it succesfully identifies this set to the questioner is another matter; we shall come back to that). Answers (2), (3) and (4) above are examples of this. An indefinite answer has a form which allows (and even suggests) that there are different sets satisfying the description it presents. Answer (5) above is an example of this. We shall focus first on definite answers. Indefinite answers will be dealt with in the next sub-section.

If it is correct, a definite minimal answer to a question describes the same object as the question. Therefore it may be represented by a logical expression which denotes the same object as the expression representing the content of the question. Here are some examples of questions and minimal definite answers with their corresponding logical expressions.

Q:	*Does John Walk?*	(6a)
	WALK(JOHN)	(6b)
A1:	*Yes.*	(7a)
	TRUE	(7b)
A2:	*No.*	(8a)
	FALSE	(8b)
Q:	*Who walks?*	(9a)
	$\{x \mid \text{WALK}(x)\}$	(9b)
A1:	*The boys.*	(10a)
	BOYS	(10b)
A2:	*John and Peter*	(11a)
	{JOHN, PETER}	(11b)

Note that neither in the case of a yes/no-question, nor in the case of a wh-question, does a minimal answer independently express a proposition. Although in the case of a yes/no-question a minimal definite answer describes a truthvalue, it must do so by means of a logical constant; in the case of a wh-question a minimal definite answer describes a set of entities. In both cases, the proposition expressed by a minimal definite answer with content A is constructed by taking into account the question with content Q which

provided the context for it: it is the proposition $Q = A$.[9] For instance, the proposition expressed by (7) as an answer to (6) is

$$\text{WALK(JOHN)} = \text{TRUE}, \tag{12}$$

the proposition expressed by (8) as an answer to (6) is

$$\text{WALK(JOHN)} = \text{FALSE}, \tag{13}$$

the proposition expressed by (10) as an answer to (9) is

$$\{x \mid \text{WALK}(x)\} = \text{BOYS}, \tag{14}$$

the proposition expressed by (11) as an answer to (9) is

$$\{x \mid \text{WALK}(x)\} = \{\text{JOHN}, \text{PETER}\}. \tag{15}$$

An answer with content A may now be defined to be *correct* if it is given in answer to a question with content Q, and $Q = A$ is true in the interpretation of the logical language corresponding to the actual world.

It is clear from the above examples that an answer is only counted as correct if it is *complete*. For instance, if, in addition to John and Peter, Harry walks as well, (11) is counted as a false answer to (9). This means that the treatment of wh-questions presented so far treats only complete answers to "mention-all" readings. In the next section, partial answers to wh-questions will be introduced.

[9] For the case of negative questions, the negation is treated as part of the illocutionary force of such questions instead of having it inside their propositional content. For instance, assuming the illocutionary force operator NEGATIVE-QUESTION for negated questions and ANSWER for minimal answers, analyses of the following form result:

Q: *Doesn't John walk?*
NEGATIVE-QUESTION (WALK(JOHN))

A1: *Yes (he does)*
ANSWER (TRUE)

A2 *No (he doesn't)*
ANSWER (FALSE)

If ANSWER's in the context of NEGATIVE-QUESTION's are now treated the same way as in the context of regular QUESTION's, the answer "Yes" results in the proposition WALK (JOHN) = TRUE, while the answer "No" results in the proposition WALK (JOHN) = FALSE.

5.4. **Indefinite Answers.**

The previous subsection only treated definite answers. There are also allowable answers, however, which lack definite reference.
A question like

Who did you bring? (1)

may be correctly (and informatively) answered by

Two Hungarian linguists (2)

without further identifying the individual items that were brought.

The treatment of the previous section can be modified so as to accommodate answers of this kind, in the following way. Indefinite answers are represented as sets. For instance, (2) is represented as

$$\boldsymbol{P}_2(\{x \in \text{LINGUISTS} \mid \text{HUNGARIAN}(x)\}) \quad (3)$$

(Notation: $\boldsymbol{P}_n(A)$ stands for the set of subsets of A which have cardinality n.) To put definite answers on the same level, they are represented as singleton sets. For instance, (4) is rendered as (5):

John and Peter (4)
$\{\{\text{JOHN}, \text{PETER}\}\}$ (5)

Thus, an answer to a wh-question identifies a set of objects and expresses that the object described by the question is one of these. The proposition expressed by an answer with content A given in reply to a question with content Q is defined as: $Q \in A$. The definition of correctness is modified accordingly: a question with content Q is correctly answered by an answer with content A iff $Q \in A$.

Multiple noun phrase answers.

Answers consisting of multiple noun phrases fit the same treatment. Disjunction between noun phrases is naturally rendered by the union operation. For instance:

John or two of his girlfriends. (6a)
$\cup(\{\text{JOHN}\}, \boldsymbol{P}_2(\text{GIRLFRIENDS}(\text{JOHN})))$ (6b)

Conjunction between noun phrases is rendered by means of the Cartesian product. For instance:

Two dogs, three girls and John. (7a)

(for: $(\boldsymbol{P}_2(\text{DOGS}) \times \boldsymbol{P}_3(\text{GIRLS}) \times \{\text{JOHN}\})$, *apply:* $(\lambda u: \textbf{set}(u))$ (7b)

where $\times$ is the operator which forms the Cartesian product, and **set** is an operator which, applied to an n-tuple, yields the set consisting of the elements of the n-tuple.

Definite noun phrases as partial answers.

Wh-questions such as:

Who walks? (8)

may sometimes be answered by

John and Peter (9)

when the answer is not intended to be an exhaustive list. This phenomenon can be described by introducing a second interpretation of definite noun phrases. In addition to the reading

$$\{\{\text{JOHN}, \text{PETER}\}\} \tag{10}$$

(9) may also be assigned the reading

$$\{X \mid \{\text{JOHN}, \text{PETER}\} \subseteq X\} \tag{11}$$

Similarly, an indefinite noun phrase may express a partial answer:

Two Hungarian linguists (12)

is then not only rendered as

$$\boldsymbol{P}_2(\{x \in \text{LINGUISTS} \mid \text{HUNGARIAN}(x)\}) \tag{13}$$

(in the exhaustive reading), but also as

$$\{X \mid \exists Y \in \boldsymbol{P}_2(\{x \in \text{LINGUISTS} \mid \text{HUNGARIAN}(x)\}): Y \subseteq X\} \tag{14}$$

In this way, the fact that the same wh-question may be correctly answered by complete answers and by partial answers may be accounted for.[10] In this approach, wh-questions are always viewed as asking for exhaustive answers (except in the case of "quantifying in" – see section 6 below).

5.5. **Full-Sentence Answers.**

In the treatment just sketched, the content of a minimal answer can be determined directly without any kind of reference to the syntactic/semantic structure of the question. The proposition expressed by the minimal answer in the context of the question is then constructed in a completely compositional way by combining the content of the answer with the content of the question.

With full-sentence answers, on the other hand, the situation is a little more complicated. Full-sentence answers often present a minimal answer embedded in a partial repetition of the question. Our treatment would involve "extracting" the minimal answer out of such questions.

Consider, for instance, the question

Which back-end processors did Akzo buy? (1)

which is represented as

$$\{x \in \text{BEPS} \mid \text{BUY}(\text{AKZO}, x)\} \tag{2}$$

The answer

Akzo bought six INTEL chips. (3)

is represented as[11]

$$\boldsymbol{P}_6(\text{INTEL-CHIPS}) \tag{4}$$

[10] Answers may be ambiguous between a complete and an incomplete reading. In spoken language, disambiguation seems to be often accomplished by intonation: a final falling intonation contour indicating a closed (i.e. complete) answer and the definite absence of closing markers indicating an open (i.e. incomplete) answer.

[11] Analysing a full-sentence answer thus involves assessing how its syntactic structure and its constituents match those of the question-sentence, so that the minimal answer which is embedded within the answer-sentence may be "extracted". Since semantic rather than syntactic matters are focussed on throughout this book, no proposals about the precise details of this correspondence between question-sentences and answer-sentences will be given.

Combining the answer-expression (4) with the question-expression (2) in the usual way, yields the following expression for the proposition communicated by answer (3) in the context of question (1):

$$\{x \in \text{BEPS} \mid \text{BUY}(\text{AKZO}, x)\} \in \boldsymbol{P}_6(\text{INTEL-CHIPS}) \tag{5}$$

It is not possible to construct (5) as a reading of sentence (3) without taking the question-context into account. To view answers as isolated propositions is therefore not a viable strategy.

To take another example: the answer to the question

Who stole my bicycle? (6)

may not only be

John. (7)

but also, correctly but less helpfully:

The people who stole your bicycle. (8)

or the rather bizarre sentence

The people who stole your bicycle stole your bicyle. (9)

which, though even more conspicuously uncooperative, can still function as an answer to (6). If we were to treat (9) as a proposition in its own right, however, it would be logically equivalent to:

Every dog which doesn't like cats doesn't like cats. (10)

(10), however, is clearly a non-sequitur to (6) while (9) is not. This fact can not be explained from the more proposition-oriented perspectives. It offers no problems to the perspective presented here.

6. **Quantifying into Questions.**

6.1. **Introduction.**

In all examples discussed so far, the illocutionary act of asking a question could be represented as ASK(C), where ASK is the operator indicating the illocutionary force, and C is an expression representing the semantic content of the question (a proposition in the case of yes/no questions, a function from states of the world to sets of individuals in the case of wh-questions). The illocutionary force operator was "bracketed out" during most of the discussion so far, since this discussion focussed entirely on the contents of questions.

Though it is often assumed that this "division of labor" between illocutionary force and semantics is generally valid, there exist counter-examples which complicate the situation. In the present section these shall be dealt with.

Wh-questions display a kind of quantifier scope ambiguity which does not seem to arise for assertions and which cannot be accounted for in the above scheme. An example of this is shown by the question

What is the price of each of Akzo's computers? (1)

if we consider the reading which requests for each of Akzo's computers a specification of its price.

6.2. **The "Compound Speech Act" Analysis.**

It has been suggested before, that the speech act of "asking" need not be viewed as an elementary action, but may be analysed in terms of the illocutionary force of "requesting" and a predicate which describes what the addressee of the question is expected to do with the question content. Applying such an analysis creates the "space" which is necessary to express the quantifier scope ambiguity in questions like (1) above.[12)]

The ASK-operator above may thus be split into two parts: an operator BRING-ABOUT, applicable to expressions of the logical language, and a predicate IDENTIFIED which is true for an individual iff it is being identified. Instead of ASK *(C)* we now get BRING-ABOUT ($\hat{}$IDENTIFIED($\check{}$*C*)). For instance

Which boys walk? (2)

is represented not as

$$\text{ASK}(\hat{}\{x \in \text{BOYS} \mid \text{WALK}(x)\}) \tag{3}$$

but as

$$\text{BRING-ABOUT}(\hat{}\text{IDENTIFIED}(\{x \in \text{BOYS} \mid \text{WALK}(x)\})) \tag{4}$$

The compound structure of the illocutionary force operator now creates the possibility to represent the quantifier scope ambiguity that seems to be present in question sentences like (1). Sentence (1) above may now be assigned two readings: the implausible one which assumes that all Akzo's computers have the same price is rendered as

[12)] Grosz (1982) suggests applying this idea to an analysis of "asking" proposed by Cohen and Perrault (1979).

$$\text{BRING-ABOUT}(\hat{}\,\text{IDENTIFIED}\ ([\iota y \mid \forall x \in \text{AKZO-COMPUTERS}:\ y = \text{PRICE}(x)])) \tag{5}$$

and the more plausible one which asks for a list of prices is rendered as

$$\text{BRING-ABOUT}\ (\hat{}\,\forall x \in \text{AKZO-COMPUTERS}:\ \text{IDENTIFIED}\ (\text{PRICE}(x))) \tag{6}$$

The sentences which led Belnap to include the "size-specification" feature in the question-operator of his erotetic system (Belnap and Steel, 1976, section 1.31) display a special case of the phenomenon treated this way. Formulated most unambiguously, these questions take the form "What's an example of a...", "What are some of the...", "What are at least three...", etc.

These cases can be dealt with along the same lines as (1) above. For instance,

What is the price of two of Akzo's computers? (7)

has two readings, which can be rendered in a way which is exactly analogous to (5)-(6) above:

$$\text{BRING-ABOUT}\ (\hat{}\,\text{IDENTIFIED}\ ([\iota y \mid \exists_2 x \in \text{AKZO-COMPUTERS}:\ y = \text{PRICE}(x)])) \tag{8}$$

$$\text{BRING-ABOUT}\ (\hat{}\ \exists_2 x \in \text{AKZO-COMPUTERS}:\ \text{IDENTIFIED}\ (\text{PRICE}(x))) \tag{9}$$

6.3. **A Proposal by Groenendijk and Stokhof.**

An ingeneous treatment along different lines was put forward in Groenendijk and Stokhof's (1981) proposal for the semantics of embedded wh-clauses, that we discussed in section 2.4 of this chapter, in which question (1) is analyzed as

$$(\lambda w:\ (\lambda i:\ \forall x \in \text{AKZO-COMPUTERS}:\ \mathbf{ext}_i\ (\text{PRICE}(x)) = \mathbf{ext}_w\ (\text{PRICE}(x)))) \tag{10}$$

(Notation: w and i range over possible worlds; $\mathbf{ext}_j\ (F)$, where j has the type "possible world", stands for $F(j)$, and yields the extension of F in j).
To understand that this is in fact correct, it may help to realize that, as Groenendijk and Stokhof point out, the universal quantification can be seen

as an abbreviation for a conjunction. (10) is then equivalent to

$$\begin{aligned}(\lambda w\text{: }(\lambda i\text{: }\ & \mathbf{ext}_i\,(\text{PRICE}\,(c_1)) = \mathbf{ext}_w\,(\text{PRICE}\,(c_1)) \ \&\\ & \mathbf{ext}_i\,(\text{PRICE}\,(c_2)) = \mathbf{ext}_w\,(\text{PRICE}\,(c_2)) \ \&\\ & \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ \&\\ & \mathbf{ext}_i\,(\text{PRICE}\,(c_n)) = \mathbf{ext}_w\,(\text{PRICE}\,(c_n)))),\end{aligned} \tag{11}$$

where C_1, ..., C_n are the proper names of all Akzo's computers.

This treatment works only for the case of "each" however, because it exploits a particular property of universal quantification which does not apply to other quantifiers. For instance, if one would try to treat

What is the price of one of Akzo's computers? (12)

along the same lines, one would get

$$(\lambda w\text{: }(\lambda i\text{: }\exists x \in \text{AKZO-COMPUTERS: }\mathbf{ext}_i\,(\text{PRICE}\,(x)) = \mathbf{ext}_w\,(\text{PRICE}\,(x)))) \tag{13}$$

Since the existential quantifier can be viewed as an abbreviation for a disjunction, (13) is equivalent to

$$\begin{aligned}(\lambda w\text{: }(\lambda i\text{: }\ & \mathbf{ext}_i\,(\text{PRICE}\,(c_1)) = \mathbf{ext}_w\,(\text{PRICE}\,(c_1)) \ \vee\\ & \mathbf{ext}_i\,(\text{PRICE}\,(c_2)) = \mathbf{ext}_w\,(\text{PRICE}\,(c_2)) \ \vee\\ & \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ \vee\\ & \mathbf{ext}_i\,(\text{PRICE}\,(c_n)) = \mathbf{ext}_w\,(\text{PRICE}\,(c_n))))\end{aligned} \tag{14}$$

Thus, Groenendijk and Stokhof's proposal lacks the generality that one should hope for.

6.4. **The PHLIQA1 Treatment.**

The present subsection present an alternative treatment of the problem described in section 6.1. This treatment, which has attractive computational properties, was employed in the PHLIQA1 system. It embodies a different analysis of the speech act of asking than the one presented in subsection 6.2.

As a point of departure, let us consider again the analysis of the contents of questions that was presented so far: the content of a yes/no question is a function from possible worlds to truthvalues, and the content of a wh-question is a function from possible worlds to sets of individuals. Questions are now viewed as displaying pragmatic objects, to be called "kernel questions", which have a one-to-one correspondence with their contents as

just described. The function QC assigns to any function φ the kernel question QC (φ) which has φ as its content.

This view is not incompatible with the speech act perspective on discourse. An illocutionary force operator DISPLAY may be assumed which operates on the question object as just described.[13] Thus,

Which boys walk? (15)

is analysed as

$$\text{DISPLAY}\,(\text{QC}\,(\hat{}\{x \in \text{BOYS} \mid \text{WALK}(x)\})) \tag{16}$$

Example (1) above is now dealt with by allowing one question sentence to display not only a single kernel-question, but a set of kernel questions as well. Thus, (1) is analysed as

$$\text{DISPLAY}\,(\textit{for:}\ \text{AKZO-COMPUTERS},\ \textit{apply:}\ (\lambda x:\ \text{QC}\,(\hat{}\{\text{PRICE}(x)\}))) \tag{17}$$

Note that nothing has to change in the account of the question-to-answer relationship as it was described before: it simply applies separately to all kernel questions in the "display set" and their answers.

There is an important difference, however, between answering to a question-sentence displaying a set of kernel questions and answering to a question-sentence displaying a single kernel question: in the former case, the short form of the answer may be less informative than the full-sentence form. For questions about prices, for instance, the short form of the answer may be simply the value of the question content.

What is the price of the Illiac? (18)

analysed as

$$\text{DISPLAY}\,(\text{QC}(\hat{}\{\text{PRICE}\,(\text{ILLIAC})\})) \tag{19}$$

is answered by the value of

$$\{\text{PRICE}\,(\text{ILLIAC})\}, \tag{20}$$

[13] Technical reformulations of results of speech act theory would be necessary, however. Many different kinds of utterances could be brought together under the common denominator of one illocutionary force called DISPLAY. What current speech act analyses account for in terms of the differences between illocutionary forces, would then be accounted for in terms of the differences between the kinds of objects displayed.

for instance

4.000.000 $ (21)

If (1) is answered in this way, the answer is a list like

512.000 $, 56.000 $, 110.000 $ (22)

It would be more informative to give answers in full-sentence form in this case. If for every element i in the extension of AKZO-COMPUTERS, the proposition expressed by

$$\{\text{PRICE}(i)\} = V, \tag{23}$$

where V is the value of PRICE(i), would be formulated in English, the questioner would not only be informed of all the prices which occur among the prices of Akzo's computers, but would also find out which computer has which price.

In order to give the same information without having to resort to full-sentence answers, PHLIQA1 does not analyse (1) as (17), but as a slightly different expression, which also asks for an identification of the elements of the question-content which are "quantified in":

DISPLAY(*for:* AKZO-COMPUTERS,
apply: (λx: QC ($\hat{}\langle x, \{\text{PRICE}(x)\}\rangle$))) (24)

PHLIQA1 answers every kernel-question by applying a function called IDENTIFICATION to the extension of its contents. Answering (24) is conveying the value of

(*for:* AKZO-COMPUTERS,
apply: (λx: IDENTIFICATION ($\langle x, \{\text{PRICE}(x)\}\rangle$))) (25)

As shown in detail in section 7.2, this results in a answer-expression like

{<<IBM, 360/20, 65 KBYTE>, $95000>,
<<PHILIPS, P1800, 240 KBYTE>, $ 312000>,
<<PHILIPS, P800, 16 KBYTE>, $ 56000>>} (26)

which identifies every computer by its brand name, type number and core size, and every price by its numerical value in dollars.

"Quantifying in" by means of indefinite quantifiers may be brought under

the scope of this treatment, if "indefinite sets of kernel questions" (represented as sets of sets of kernel questions) are allowed to be expressed by an interrogative sentence. For instance,

What is the price of two of Akzo's computers? (27)

is, in the reading we are interested in now, analysed as

DISPLAY (*for:* P_2 (AKZO-COMPUTERS),
apply: (λY: (*for:* Y,
apply: (λx: QC ($\hat{}\langle x$, {PRICE(x)}$\rangle$))))) (28)

The argument of DISPLAY here denotes a set of sets of kernel questions. In displaying such a set, the questioner requests that an arbitrary element of it be answered. (If this extension is adopted, the treatment of "quantifying in" with "each" must be adapted, as displaying a singleton set with as its element a set of kernel questions.)

7. **The Pragmatics of Answering.**

7.1. **Categories of Answers.**

It is possible and desirable to make distinctions between different kinds of complete correct answers, as the three answers (2), (3), (4) to question (1) show.

Q: *Which numbers did John write on the blackboard?* (1a)

$\{x \mid \text{JWOB}(x)\}$ (1b)

A1: *The numbers that John wrote on the blackboard.* (2a)

$\{\{x \mid \text{JWOB}(x)\}\}$ (2b)

A2: *The numbers that Mary wrote in the notebook.* (3a)

$\{\{x \mid \text{MWIN}(x)\}\}$ (3b)

A3: *Five and seventeen.* (4a)

$\{\{5, 17\}\}$ (4b)

While answer A1 is certainly correct, it is certainly never the desired answer. This may be explained by pointing out that it is *uninformative* – an answer with content A in reply to a question with content Q may be defined to be uninformative iff Q ∈ A is logically equivalent to TRUE.[14)]

A2 and A3 are both informative answers, according to this definition. Which one is preferred, depends on the context in which the question is asked. Though A3 is in a sense more explicit, A2 may sometimes be more interesting.

A3 has a special property which is worth considering: it identifies the set of individuals the question asked about, without relying on any non-linguistic knowledge on the part of the hearer. Expression (4b) is an *L-determinate* expression (Carnap, 1947): an expression which has the same denotation for all interpretations of the language. The natural language formulation of this answer used the logical proper names which the English language contains for the individuals involved: "five" and "seventeen". For individuals belonging to a non-mathematical type, however, natural languages usually do not have logical proper names. L-determinate answers are therefore not always possible. The general case is, that an answer must rely on a certain amount of knowledge concerning contingent facts on the part of the questioner, in order to identify to him the individuals he asked about.

It may be observed that an adequate answer must give information about the identity or the properties of the individuals in the set described by the question. New information which only says something about the set as a whole is not enough. If (5) is answered by (6) or (7), for instance, a feeling of somewhat evasive behaviour on the part of the answerer results.

Which IBM computers did you buy last year? (5)
Some computers. (6)
Three IBM computers. (7)

[14)] This looks as if correct answers to mathematical inquiries are necessarily uninformative. This need not be the case, however, if the mathematical terms of natural language are translated into descriptive expressions, rather than logical ones. The mathematical metalanguage used to describe the semantics of English must be kept "inaccessible" for its objects: natural language expressions cannot *refer* to the notions underlying the formalism that is used for explicating their meaning. Therefore, for instance, in

Does two plus two equal four?

"two", "four", "plus" and "equal" must be analysed as expressions with descriptive types. The definitions of mathematical notions must be introduced as contingent truths, and the "necessary truth" of mathematical propositions be explicated by quantifying over a subset of the possible worlds.

A treatment along these lines may avoid the separation between mathematical and empirical propositions we find in Tichý (1978ab).

7.2. **Pragmatic Strategies.**

From this discussion, it should be clear that semantic considerations alone do not determine what an adequate answer to a given question is. Semantic considerations do impose important boundary conditions: an answer must at least be correct and informative. Whether a wh-question asks for an identification of separate individuals or for a characteristic property of a set, however, is a different matter which cannot be decided by considering the linguistic form of the question only but depends on the non-verbal context in which the question is asked. If an identification of individuals is what is desired, a model of the epistemic state of the questioner is needed if one is to guarantee that the identification will be succesful.

For a computer question answering system, a complete model of the epistemic state of the questioner is obviously not available. But a system which does not answer isolated questions but conducts longer "dialogues" may gather useful information about the questioner's knowledge and interests from the "history" of the dialogue. The dimensions of the subject domain that are used to describe the objects in questions are probably different ones than those that should be used in the answers. Follow-up questions triggered by a similar previous question indicate what the dimensions are that the questioner *is* interested in. So far, succesfully implemented systems do not use dialogue history in this way, however. They use a different strategy: relying on plausible general assumptions.

For instance, a system which answers "exam questions" about its model of a visual scene which at the same time is displayed on a screen may identify objects in this scene by means of definite descriptions in terms of the properties displayed on the screen. The answers given by the SHRDLU system (Winograd, 1972) belong to this category. The problem of constructing the appropriate definite descriptions was studied more thoroughly in the context of the HAM-RPM system (Wahlster et al., 1978).

For a system like HAM-RPM, two phases can be clearly distinguished within the process of answering a wh-question about the objects in a visual scene. The first involves finding the objects described by the question, i.e. constructing an internal representation of this set of objects in terms of logical proper names. In the second phase it is decided how to identify these objects to the questioner, in terms of the visual features of the scene, in a way which is informative and maximally efficient.

For the practice of question-answering, indefinite answers are important because very often the objects that a question asks about cannot be uniquely identified. In that case, it may nevertheless be possible to give a satisfying indefinite answer.

The fact that the information which would uniquely identify an object is

lacking in a data base is not always an accident. Often, those who query a data base about certain kinds of objects are not so much interested in the identity of these objects, as in certain of their properties. Two examples of data bases where this situation is quite common are the REL data base about ships and their cargo, and the PHLIQA1 data base, about computers and their users.[15)]

REL gives for certain kinds of objects no property that could possibly identify it, but only its category. For instance, the question

What is on the upper vehicle storage area of the USS Ogden? (1)

receives the answer

torpedoes
torpedoes
torpedoes
torpedoes
torpedoes
torpedoes
torpedoes
torpedoes
torpedoes
torpedoes (2)

which means "ten torpedoes".

In the PHLIQA1 program, some salient attributes are distinguished for every kind of object. As an answer to a question about objects of a given kind, the values of these attributes are given for every object in the answer-set. For instance, the question

Which computers did Akzo buy? (3)

would be represented as something like

$$\{x \in \text{COMPUTERS} \mid \text{BUY}(\text{AKZO}, x)\} \tag{4}$$

For a given data base, the evaluation of (4) (i.e. its transformation into a simplest equivalent expression in terms of logical proper names; see Chapter IV, section 2) may yield

$$\{C_{10}, C_{12}, C_{48}\} \tag{5}$$

where C_{10}, C_{12} and C_{48} are logical proper names for individual computers. These logical proper names have no natural language equivalents – and if

[15)] REL is described in Henisz-Thompson and Thompson (1978). The above example is from a "live" session I conducted with the system on August 17, 1979.

they had, this would not be what the questioner would be interested in. Therefore, the PHLIQA1 system does not directly evaluate the expression which represents the question content. First, the function IDENTIFICATION is applied to it. This function expresses how different kinds of entities may be described to the questioner. Depending on the type of its argument, it is further translated into an expression which indicates in detail how the elements of the "question-set" will be described. For instance, (4) is first transformed into

$$\begin{array}{l}(\textit{for:}\ \{x \in \text{COMPUTERS} \mid \text{BUY}(\text{AKZO}, x)\},\\ \ \textit{apply:}\ \text{IDENTIFICATION})\end{array} \tag{6}$$

and when the type of the arguments of the application of IDENTIFICATION is taken into account, this is further translated into

$$\begin{array}{l}(\textit{for:}\ \{x \in \text{COMPUTERS} \mid \text{BUY}(\text{AKZO}, x)\},\\ \ \textit{apply:}\ (\lambda y:\ \langle \text{BRAND}(y),\\ \qquad\qquad \text{TYPE}(y),\\ \qquad\qquad \text{CORESIZE}(y)\rangle))\end{array} \tag{7}$$

which is, for the data base just considered, equivalent to

$$\begin{array}{l}(\textit{for:}\ \{C_{10}, C_{12}, C_{48}\}\\ \ \textit{apply:}\ (\lambda y:\ \langle \text{BRAND}(y),\\ \qquad\qquad \text{TYPE}\ (y),\\ \qquad\qquad \text{CORESIZE}(\text{Y})\rangle))\end{array} \tag{8}$$

and to

$$\begin{array}{l}\{\langle \text{BRAND}(C_{10}), \text{TYPE}(C_{10}), \text{CORESIZE}\ (C_{10})\rangle,\\ \ \langle \text{BRAND}(C_{12}), \text{TYPE}(C_{12}), \text{CORESIZE}(C_{12})\rangle,\\ \ \langle \text{BRAND}(C_{48}), \text{TYPE}(C_{48}), \text{CORESIZE}(C_{48})\rangle\}\end{array} \tag{9}$$

Thus, if instead of (4), (7) is evaluated, the result might be:

$$\begin{array}{l}\{\langle \text{IBM}, 360/20, 65\ \text{KBYTE}\rangle,\\ \ \langle \text{PHILIPS}, \text{P1800}, 240\ \text{KBYTE}\rangle,\\ \ \langle \text{PHILIPS}, \text{P800}, 16\ \text{KBYTE}\rangle\}\end{array} \tag{10}$$

Note that the simple way in which PHLIQA1 gives its indefinite answers is only applicable within certain limitations. The comprehensibility of the answer relies on the fact that the questioner can often infer from a value

(such as "IBM", "360/20", "65 kbyte"), which aspect of a computer this value describes (i.e. brand, type, coresize). This is not necessarily always the case. A more generally applicable method would also indicate in the answer the identification function that was used in obtaining the values. Such an answer might then be rendered in English as "an IBM of type 360/20 with 65 kbytes of core, a Philips of type P1800 with 240 kbytes of core and a Philips of type P800 with 16 kbytes of core".

8. **Conclusion.**

Interrogatives are natural candidates for serving as an interface medium between people and computer systems which provide information on request. An automatic data base access system with suitable deductive capabilities is designed to fulfill exactly the expectations a natural language user may have when he asks a question to a conversation partner who is obedient and conscientious, although unimaginative and literal-minded. Natural language question-answering systems of this sort are the subject of this book.

In a question-answering system with a modular structure, it is useful to make use of separate modules for understanding the incoming question and for answering it on the basis of the information in the knowledge base of the system. The question understanding module must communicate to the answer computation model the "content" of the question; a precise specification of what the desired information is. Formulating the content of questions in a precise way so that it is possible to give a model-theoretic account of the connection between the content of a question and the equally precisely described content of a correct answer is a necessary precondition, in my opinion, of performing the question answering task satisfactorily. The present chapter reviewed the literature on this topic and presented an original proposal for dealing with many of the more difficult problems in an adequate way. This approach was in large part implemented in the PHLIQA1 question-answering system. The structure and operation of this system are described in the following chapter.

Chapter III. The PHLIQA1 Question-Answering System

1. Introduction.

The PHLIQA1 question-answering system represents questions and answers as logical expressions in the fashion described in the previous chapter. In the present chapter, the structure of the PHLIQA1 system will be described while the knowledge representation methods used in the system are the topics of discussion in Chapter IV and V. Before going on with detailed descriptions of the system, the question answering task PHLIQA1 was designed to carry out will be sketched along with the structure of the CODASYL data base about which the queries were asked.

2. External Requirements for a Question-Answering System.

The present section specifies the task that the PHLIQA1 question-answering system was designed to perform. In short, the system is required to answer isolated English language questions typed in by a single user through an alphanumeric terminal. The questions inquire about the state of affairs in a limited subject domain, represented in some conventional data base. The data base is assumed to be given prior to the design of the question-answering system. It may use arbitrary kinds of storage structures. The user is not aware of the structure of the data base; he is only given an informal characterization of the subject domain that he can ask questions about. He may only ask genuine questions, in a rather strict sense of the word: requests for information, formulated in a direct and literal way, which have the syntactic form of a question. He cannot add new information, introduce temporary hypotheses, etc.

The system may give curt answers. For instance, a yes/no question may be simply answered by "yes" or "no", a "how-many"-question may be answered by a number, a "which"-question may be answered by a list of names. To get a more concrete feeling for what is involved in the design of such a system, let us consider a more or less realistic CODASYL data base and the natural language questions that can be asked about it.

The data base we shall consider contains the kind of data that might be used by the marketing department of a computer manufacturer in Holland: information about the computer installations in use in the countries of the European Common Market, and about the companies where they are installed.

The structure of the data base is depicted in *fig. 1.* The boxes stand for *record types*, indicated in capitals. The data base contains a number of

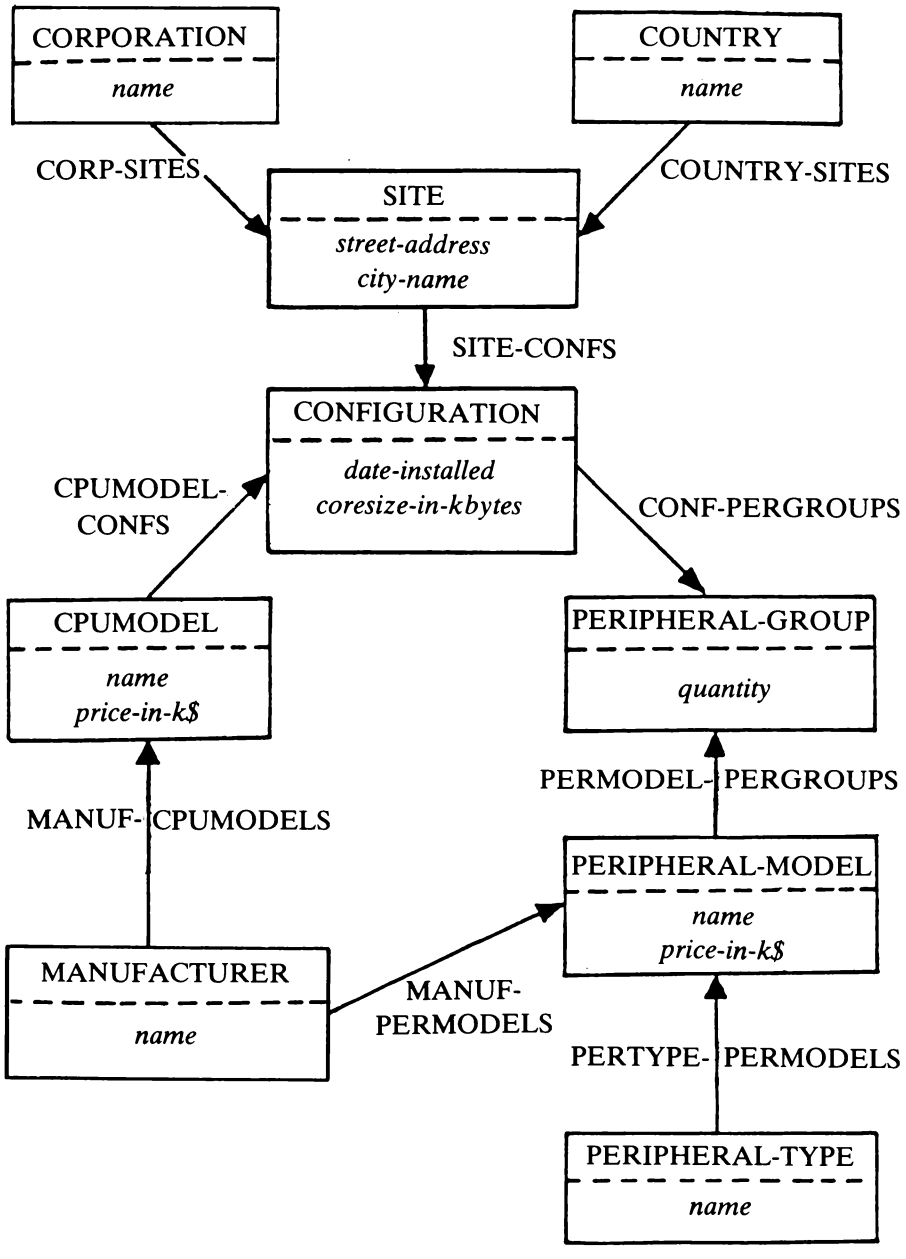

Fig. 1. Data Base Schema

records of every record type. Records correspond to real or imaginary objects. For example, every record of the type CONFIGURATION corresponds to a computer used by a European company. The two most important ways of storing information about such objects in a CODASYL data base are:

1. By means of attribute values in the record itself. For example, the attribute *date-installed* is defined for records of type CONFIGURATION. The value of this attribute represents the month and the year in which the computer was installed at its current user. Values of attributes are always strings of symbols, such as names or numbers.
2. By means of link-sets. These are indicated by arrows in fig. 1. Each arrow indicates a link-set type, defined between two record types. The record type located at the tail of the arrow is called the owner record type and the record type located at the head is called the member record type. For each occurence of the owner record type there is an occurrence of the link-set which relates the occurrence of the owner record type to zero or more occurrences of the member record type. In fig. 1, for example, each occurrence of the link-set type COUNTRY-SITES relates a COUNTRY record and a number of SITE records. This is intended to represent the relation between a country and the sites in this country where computers are installed.

In this way, the data base represents information about computer configurations, such as

- the SITE where the computer is installed, and the CORPORATION to which a SITE belongs,
- the model and the manufacturer of the central processing unit (CPU-MODEL),
- the peripheral equipment (PERIPHERAL-GROUP).

Some complications displayed by this data base are:

- Countries are represented as records, but cities are only indirectly represented by the attribute *city-name* in the record type SITE.
- Peripherals are represented by a record of type PERIPHERAL-GROUP for each group of peripherals of the same model belonging to the same computer configuration. The attribute *quantity* specifies the number of peripherals in such a group. Of course, a human interrogator of PHLIQA1 need not be aware of this and may use freely the words "peripherals", "cities" and "countries".
- We do not assume that the attribute whose values identify the different records within a record type always has some intrinsic meaning. For reasons of storage and retrieval there are unique keys for each record; although such keys may sometimes coincide with values of attributes that are of interest to the user (for example, the attribute "name" in the record type CPU-MODEL), in general this need not be the case. In order to formulate

an answer to a question, the system may therefore have to choose an appropriate way of identifying the object represented by a record.

Let us now look at some questions which one might ask about the subject domain which has its state of affairs represented by the data base of fig. 1.[1)]

1. How many computers are there in the Netherlands?
2. What is the number of IBM computers in Germany?
3. What computers in Eindhoven were installed before 1970?
4. In what month in 1972 did Shell buy a computer from IBM?
5. Does each computer in Eindhoven have a cpu made by Philips?
6. What companies have a computer with a cpu that costs more than 100.000 dollars?
7. What is the price of the most expensive configuration of Unilever?
8. How many bytes of core memory does Akzo's Arnhem computer have?
9. Are there companies with several IBM computers that have peripherals not made by IBM?
10. Which of the companies that possess more than 2 configurations bought a cpu before May '68?

Question 1 is one of the simplest questions that can be imagined. But even this question raises problems if we try to answer it on the basis of the information in the data base. The question asks for a number: the number of computers in the Netherlands. This number is not explicitly present in the data base. Assuming that records of type CONFIGURATION correspond to computers, the system will have to count the records which stand for computers in the Netherlands. However, CONFIGURATION records do not directly contain information about the country where the computer is. PHLIQA1 must use the fact that this is the country of the site where the computer is located.

Other problems are raised by words like "month", "cpu", and "companies" in questions 4, 5 and 6. Neither months nor cpus are represented in the data base, but there are codings for month and year together, and there is a record for the cpu model of every configuration. Companies are represented in the data base in two different ways: computers users as CORPORATION, computer manufacturers as MANUFACTURER. A company which is both is represented twice. Another problematic aspect of the natural language questions consists in the various kinds of ambiguity which they allow; for example, in their syntactic structure (see example questions 6 and 9) and in the meanings of their words.

These fairly arbitrary examples show a large gap between the English formulation of the questions and the manner in which the relevant information is stored in the data base.

[1)] All these questions are actually answered by the PHLIQA1 Program.

How to bridge this gap in a reliable and computationally effective way is an important design issue in the construction of a question-answering system.

3. The Top Level Design of PHLIQA1.

Constructing a logical representation of the content of a question is a useful intermediary step in the complicated process of computing an answer to a question formulated in ordinary English. Such a representation shows in a simple and unambiguous way what information the questioner wants without, however, becoming involved with the details of how to compute it. In Chapter II it was already discussed how the contents of questions as well as the contents of answers may be represented so that the correctness of an answer to a question can be accounted for. It should be clear, then, that the question answering function may be decomposed into three parts:

- the function which assigns to an input-question the logical representations of its readings.
- the function which assigns to every formal query expressed by a logical formula an adequate answer expression
- the function which assigns to an answer, represented as a logical formula, a natural language formulation.

Distinguishing between the three component functions is important, because the entire question-to-answer mapping is too complicated a function to describe in one stroke. In order to implement the mapping in an efficient and reliable manner it is necessary to break it down into as many separate components as possible. We must therefore take advantage of any conceptual distinctions which may lead us to distinguish well-defined sub-components.

For the sake of simplicity, in this treatment of PHLIQA1 a rather trivial version of the last function which translates a logical formulation of an answer into an answer in natural language will be assumed. The possibility for further subdivision of the first two functions will be considered in some detail, however.

What a logical representation of a reading of a question amounts to depends on the semantic primitives which are assumed in the logical language which is used. We shall argue that several different levels may be used in succession: an English-oriented Formal Language (EFL), a formal language which contains one descriptive constant for every descriptive lexical item of English; a World Model Language (WML) whose constants correspond to the concepts which constitute the subject domain; and the Data Base Language (DBL) whose constants are determined by the structure of the data base of the system. Assuming that these three levels are reasonable and sufficient intermediate steps between a question and its answer, the question-

to-answer function would be seen as the composition of:
- the function which assigns to any question its EFL representations.
- the function which assigns to any EFL expression its WML representations.
- the function which assigns to any WML expression its DBL representations.
- the function which assigns to any DBL expression the values it may have according to the data base.
- the function which assigns to a value expression a natural language answer formulating it.

(See *fig. 2.*) The three languages, EFL, WML and DBL will now be discussed in more detail.

4. An English-Oriented Level of Meaning Representation.

The EFL representation of an input question expresses only the aspects of its meaning that do not depend on the subject domain. These aspects include the semantic consequences of the syntactic structure of the sentence, the meaning of "function words" (such as "the", "each", "and", "than", etc.), and the semantic consequences of the internal structure of descriptive words (e.g. analysing "computers" as the plural of "computer", and "biggest" as the superlative of "big").

Without taking the subject-domain into account, however, the referential aspects of word meaning (such as the notion "computer" or the notion "big") cannot be analysed. Relations between meanings of different words never hold completely generally, but always depend on the domain of discourse to which the words are applied.

The semantic analyses which are put forward in the context of formal linguistics and philisophical logic are usually subject-domain independent in this way (see, e.g., Montague (1970, 1973) and related work). They assume one semantically primitive descriptive constant in the logical language for every descriptive lexical item of the natural language. However, if we want to allow the natural language words to be ambiguous, this method must allow more than one lexical entry for one word or morpheme. And the question: "how many entries do we need?" cannot be assessed independently of the subject domain to be addressed. Landsbergen and Scha (1977) conclude therefore that a truly subject-domain-independent meaning representation can only be formulated in terms of a logical language which is ambiguous.[2)]

[2)] It is not possible to account for the different meanings of a constant by the fact that it can have different denotations under different interpretations. This is most clearly seen when we consider a sentence which contains one word used in two meanings, such as "The pen is in the pen". If we have only one constant representing "pen", and an interpretation of the language assigning one denotation to every constant, we preclude the possibility that the two occurrences refer to different objects. See Landsbergen and Scha (1977, section 4.5), and Bennett's (1978, note 4) comments on Montague (1970, pp. 209/210).

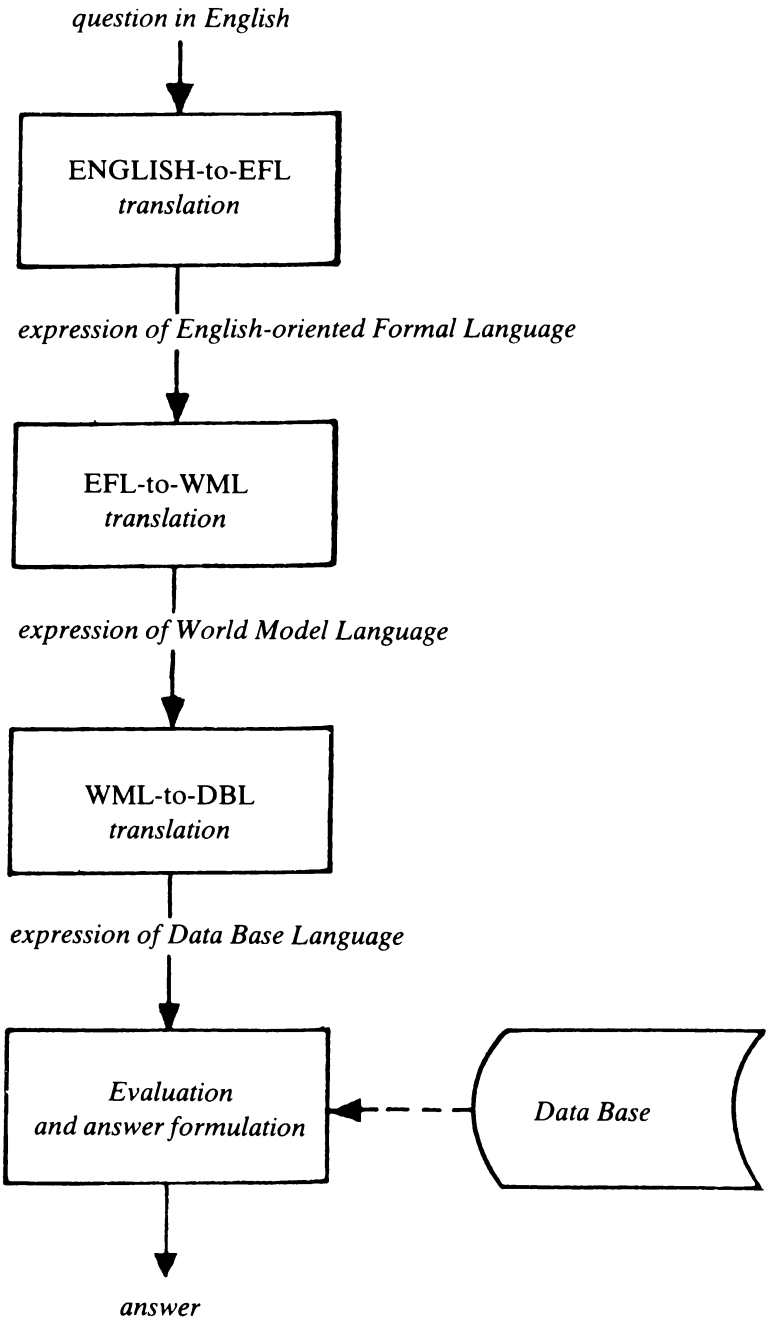

Fig. 2. A global diagram of the PHLIQA1 system

The model theory of such a "super-language" is more complicated than the model theory for an ordinary logical language. We cannot consider directly the denotation of an expression under an interpretation of the language, but we must use two steps: first defining the unambiguous **instances** of the super-language expressions, and then assigning an interpretation to the "instance language". Put somewhat more formally, the semantics of the language is defined as follows:

1. An Instance Language is defined, which is syntactically identical to the superlanguage except for its (unambiguous) constants.
2. A Constant Instantiation CI is defined: a function assigning to every constant c of the super-language a set of constants CI(c) of the Instance Language. This defines for every super-language expression e a set of instance expressions EI(e): exactly all those instance language expressions which could be generated by replacing every ambiguous constant c in e by an element of CI(c).[3)]
3. The Instance Language is interpreted in the usual way. This interpretation defines for every instance language expression a denotation, and for every super-language expression a set of denotations: the denotations of their instance expressions.

Logical equivalence and similar notions may be defined for the super-language in a rather self-evident way. EFL expressions A and B are logically equivalent if for any instantiation function EI:

$$\forall x \in \text{EI}(A): \exists y \in \text{EI}(B): x \equiv y \;\; \& \\ \forall y \in \text{EI}(B): \exists x \in \text{EI}(A): x \equiv y.$$

where we use $\equiv$ for logical equivalence between expressions of the instance language.

Using EFL as an intermediate level of representation has an important practical advantage: a well-defined meaning representation can be constructed in parallel with the syntactic parse of the input question, while the treatment of semantic word-ambiguities is postponed until after that phase. Because many words turn out to be manifold ambiguous when their meaning is analysed in a precise logical framework, this set-up is more efficient than one which puts the word-ambiguities in the lexicon.

[3)] Compared to the definition actually employed in the PHLIQA1 system, this is a little simplified. See Bronnenberg et al. (1980, section 6.1) for details of this definition. The unpleasant complexities of these details follow from the desire to avoid semantically anomalous instance expressions. (See Appendix for definition of *semantic anomaly*.) It might be preferable to allow semantically anomalous expressions and to simplify the definition accordingly.

The descriptive atomic types and descriptive constants of EFL.

In EFL we do not subdivide the domain of "real world" individuals into distinct categories. There is only one descriptive atomic type: *entity*.
For every descriptive word of English there is one constant. Homonyms with identical syntactic properties are thus represented by one constant. As we shall see below, the semantic type of the constant only depends on the syntactic category of the corresponding word.
For every proper name there is a constant of type *entity*, e.g. EINDHOVEN for the word "Eindhoven", HOLLAND for the word "Holland".
For every noun there is a constant of type S(*entity*), e.g.
CPUS for the words "cpu" and "cpus",
CONFIGURATIONS for the words "configuration" and "configurations",
COMPUTERS for the words "computer" and "computers",
CITIES for the words "city" and "cities",
P1400S for the words "P1400" and "P1400s".
For every preposition there is a constant of type
(⟨*entity, entity*⟩ → *truthvalue*),
e.g. IN for the word "in",
OF for the word "of".
For every verb there is a constant with a type of the form
(⟨*entity, ..., entity*⟩ → *truthvalue*),
depening on the number of arguments that the verb takes.
For instance:
for "to exist", EXIST with type (⟨*entity*⟩ → *truthvalue*),
for the main verb "to be", BE with type
(⟨*entity, entity*⟩ → *truthvalue*),
for the main verb "to have", HAVE with type
(⟨*entity, entity*⟩ → *truthvalue*),
for "to possess", POSSESS with type
(⟨*entity, entity*⟩ → *truthvalue*).
For every adjective there is a constant of type
(*entity* → *truthvalue*), e.g. DUTCH for "Dutch".

5. **The World Model Language.**

The *World Model Language* is a formal language whose constants correspond to concepts which form a set which characterizes PHLIQA1's subject domain. Its expressions have different "semantic types"; they can denote truth values, various other kinds of individual objects (e.g. cpus or integers), collections (e.g. sets or lists), functions, etc.

A set of concepts is defined as characterizing a certain subject domain, if it has the following properties:

- If one knows the extension of each of the concepts, one is completely informed about the state of affairs in the subject domain.
- The extension of each of the concepts is independent of the extensions of all the other concepts.[4)]

The first property establishes that a "characterizing set" includes enough concepts to cover all aspects of the subject domain. The second property establishes that this happens in a parsimonous way: the set does not include unnecessarily many concepts. In particular, it does not include concepts which are definable in terms of other concepts which are included. For example, a subject domain may involve the relation between a site and the city where it is located, as well as the relation between a city and its country. In that case the "characterizing set" does not include the relation between a site and its country, since this relation is definable in terms of the other relations.

For the subject domain of PHLIQA1, the descriptive atomic types and descriptive constants of WML are indicated below.

Descriptive atomic types:

company, *site*, *country*, *city*, *conf* (for configuration), *cpu*, *cpumodel*, *periph* (for peripheral), *per-model* (for peripheral model), *per-type* (for kind of peripheral), *cmem* (for core memory), *calmonth* (for calender month), *calyear* (for calendar year), *mem-unit* (for memory unit), *money-unit*, *dur-unit* (for duration unit).

Descriptive constants.

For some of the atomic types there are descriptive constants. We shall not list these, but only give some examples.
For type *company:* PHILIPS, IBM, AKZO.
For type *country:* NETHERLANDS, BELGIUM, FRANCE.
For type *city:* EINDHOVEN, AMSTERDAM, PARIS.
For type *calmonth:* JANUARY, FEBRUARY.
For type *calyear:* Y1960, Y1961.
For type *cpumodel:* P1400.

[4)] In data base terminology, a World Model Language as defined here is a *completely normalized data model without functional dependencies*. It may not always be possible to formulate such a data model for any given subject domain in a given logical language. Dependencies must then be formulated explicitly by means of axioms. See Van Griethuysen, 1982.

For every atomic type α there is a constant of type $S(\alpha)$ which denotes, under every interpretation, the domain of α. This constant is written as: GS_{α}. For instance: $GS_{company}$, GS_{site}, etc.
The other descriptive constants are functions:

Function	*Type*
F-CY-NAME	(*company* → *string*)
F-COUNTRY-NAME	(*country* → *string*)
F-CITY-NAME	(*city* → *string*)
F-CPUMODEL-NAME	(*cpumodel* → *string*)
F-PERMODEL-NAME	(*per-model* → *string*)
F-PERTYPE-NAME	(*per-type* → *string*)
F-CALMONTH-NAME	(*calmonth* → *string*)
F-SITE-ADDRESS	(*site* → *string*)
F-SITE-CITY	(*site* → *city*)
F-SITE-COMPANY	(*site* → *company*)
F-CITY-COUNTRY	(*city* → *country*)
F-CPUMODEL-SITE	(*cpumodel* → *site*)
F-PERMODEL-SITE	(*per-model* → *site*)
F-CONF-SITE	(*conf* → *site*)
F-CMEM-SIZE	(*cmem* → *AMT (mem-unit)*)
F-CMEM-CONF	(*cmem* → *conf*)
F-CPU-CONF	(*cpu* → *conf*)
F-PERIPH-CONF	(*periph* → *conf*)
F-CPU-MONTH-INST	(*cpu* → *calmonth*)
F-CPU-YEAR-INST	(*cpu* → *calyear*)
F-PERIPH-MONTH-INST	(*periph* → *calmonth*)
F-PERIPH-YEAR-INST	(*periph* → *calyear*)
F-CPU-CPUMODEL	(*cpu* → *cpumodel*)
F-PERIPH-PERMODEL	(*periph* → *permodel*)
F-PERMODEL-PRICE	(*permodel* → *AMT (money-unit)*)
F-CPUMODEL-PRICE	(*cpumodel* → *AMT (money-unit)*)
F-PERMODEL-PERTYPE	(*permodel* → *pertype*)
F-CALMONTH-NR	(*calmonth* → *integer*)
F-CALYEAR-NR	(*calyear* → *integer*)
F-CMEM-PRICE	(*cmem* → *AMT (money-unit)*)

The functions that have a type of the form $(\alpha \rightarrow \beta)$, where β is a type other than *string*, are shown in *fig. 3*. The boxes represent types, and an arrow pointing from a box labelled α to a box labelled β represents a function of type $(\alpha \rightarrow \beta)$.

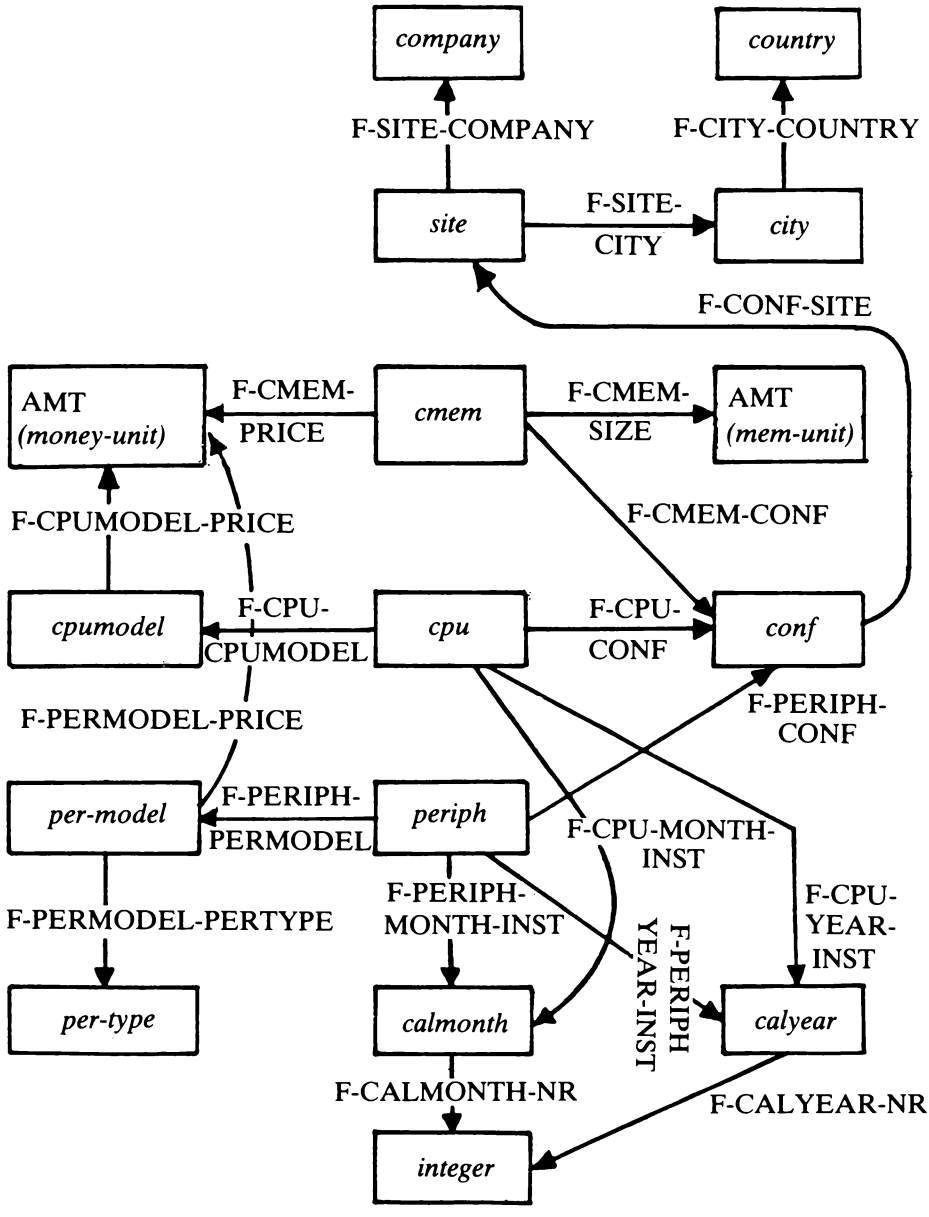

Fig. 3. WML function constants and the types of their domains and ranges

6. The Data-Base Language.

Just like the English-to-WML translation, the computation of an answer on the basis of a WML expression is divided into two distinct steps: The WML expression is translated into an expression of a language called the *Data Base Language* (DBL), and the answer is computed by evaluating the DBL expression.

DBL has constants which correspond to the data base primitives, i.e. to the various record types, attributes, etc. (see section 2). A DBL expression shows explicitly how the answer to the question depends on the information in the data base.

Although the subject domain is largely determined by the data base, the primitive notions that characterize the subject domain will generally not coincide with the data base primitives, which are chosen with an eye to the efficiency of storage and retrieval of information. Therefore, the WML and DBL languages are different. For instance "cpu", "city" and "year" are among the concepts that belong to the subject domain of PHLIQA1 without corresponding to data base primitives.

A CODASYL data base is a specification of the extensions of record-types, attributes and link-sets. This can easily be translated into standard mathematical terminology.[5)]

A *record type* is a set of individuals.

An *attribute* is a function which has one of the record types as its domain of application, and which has as its range a subset of the strings or the integers.

A *link-set* is a function which has a record type as its domain, and another record type as its range. (The CODASYL implementation of the specification of the extension of this function also makes the extension of its inverse immediately available).

In the PHLIQA1 data base, for instance, the link-set COUNTRY-SITES specifies a function F-SITE-COUNTRY, from site records to country records (and its inverse F-COUNTRY-SITES, from country records to sets of site records). For each site record S there is a link-set occurrence of the link-set COUNTRY-SITES which has S as a MEMBER; the country record G which is the OWNER of this link-set occurrence is the value of the function F-SITE-COUNTRY for the argument S. (Similarly, for any given country record G, there is a link-set occurrence which has G as its OWNER; the set of MEMBERS of this link-set occurrence is the value of F-COUNTRY-SITES for the argument G). It is clear that, given a CODASYL data base, the descriptive atomic types and the descriptive constants of a corresponding Data Base Language can be derived in a systematic way: For every record type R we have a descriptive atomic type α_R

[5)] See Chapter IV, section 4, for a more detailed discussion.

(and, as always, we have for every referential atomic type τ a constant GS_τ denoting its domain).

For every attribute of record type R, we have a function constant of type $(\alpha_R \rightarrow \textit{string})$ or $(\alpha_R \rightarrow \textit{integer})$, depending on the kind of values of the attribute. For every link-set that has R as its OWNER record type and M as its MEMBER record type, we have a function of type $(\alpha_M \rightarrow \alpha_R)$, and its inverse, a function of type $(\alpha_R \rightarrow S(\alpha_M))$.

In this way we derive the types and constants of DBL from the CODASYL declaration of the PHLIQA1 data base, described in section 2 (fig. 1). This results in the following types and constants:

Descriptive atomic types:

$\textit{corporation}_D$, $\textit{site}_D$, $\textit{configuration}_D$, $\textit{cpu-model}_D$, $\textit{country}_D$, $\textit{manufacturer}_D$, $\textit{peripheral-group}_D$, $\textit{peripheral-model}_D$, $\textit{peripheral-type}_D$.

Descriptive constants (apart from the generic constants for the descriptive atomic types):

F-CORP-NAME$_D$ with type $(\textit{corporation}_D \rightarrow \textit{string})$
F-COUNTRY-NAME$_D$ with type $(\textit{country}_D \rightarrow \textit{string})$
F-SITE-STREETADDRESS$_D$ with type $(\textit{site}_D \rightarrow \textit{string})$
F-SITE-CITYNAME$_D$ with type $(\textit{site}_D \rightarrow \textit{string})$
F-CONF-DATEINST$_D$ with type $(\textit{conf}_D \rightarrow \textit{integer})$
F-CONF-CORESIZE$_D$ with type $(\textit{conf}_D \rightarrow \textit{integer})$
F-PERGROUP-QUANTITY$_D$ with type $(\textit{peripheral-group}_D \rightarrow \textit{integer})$
F-PERMODEL-NAME$_D$ with type $(\textit{peripheral-model}_D \rightarrow \textit{string})$
F-PERMODEL-PRICE$_D$ with type $(\textit{peripheral-model}_D \rightarrow \textit{integer})$
F-PERTYPE-NAME$_D$ with type $(\textit{peripheral-type}_D \rightarrow \textit{string})$
F-CPUMODEL-NAME$_D$ with type $(\textit{cpu-model}_D \rightarrow \textit{string})$
F-CPUMODEL-PRICE$_D$ with type $(\textit{cpu-model}_D \rightarrow \textit{integer})$
F-MANUFACTURER-NAME$_D$ with type $(\textit{manufacturer}_D \rightarrow \textit{string})$
F-SITE-CORP$_D$ with type $(\textit{site}_D \rightarrow \textit{corporation}_D)$
F-SITE-COUNTRY$_D$ with type $(\textit{site}_D \rightarrow \textit{country}_D)$
F-CONF-SITE$_D$ with type $(\textit{configuration}_D \rightarrow \textit{site}_D)$
F-CONF-CPUMODEL$_D$ with type $(\textit{configuration}_D \rightarrow \textit{cpumodel}_D)$
F-PERGROUP-CONF$_D$ with type $(\textit{peripheral-group}_D \rightarrow \textit{configuration}_D)$
F-PERGROUP-PERMODEL$_D$ with type $(\textit{peripheral-group}_D \rightarrow \textit{peripheral-model}_D)$
F-PERMODEL-PERTYPE$_D$ with type $(\textit{peripheral-model}_D \rightarrow \textit{peripheral-type}_D)$
F-PERMODEL-MANUF$_D$ with type $(\textit{peripheral-model}_D \rightarrow \textit{manufacturer}_D)$
F-CPUMODEL-MANUF$_D$ with type $(\textit{cpu-model}_D \rightarrow \textit{manufacturer}_D)$

For each function *F-ALFA-BETA*$_D$ with a type ($\alpha \rightarrow \beta$) where β is not *string* or *integer*, the inverse function *F-BETA-ALFAS*$_D$ with type ($\beta \rightarrow S(\alpha)$) is also part of DBL.
Those functions which have a type ($\alpha \rightarrow \beta$) where β is a descriptive type of DBL are shown in *fig. 4*. The boxes represent types. An arrow pointing from a box labelled α to a box labelled β represents a function constant with type ($\alpha \rightarrow \beta$) and its inverse, with type ($\beta \rightarrow S(\alpha)$).

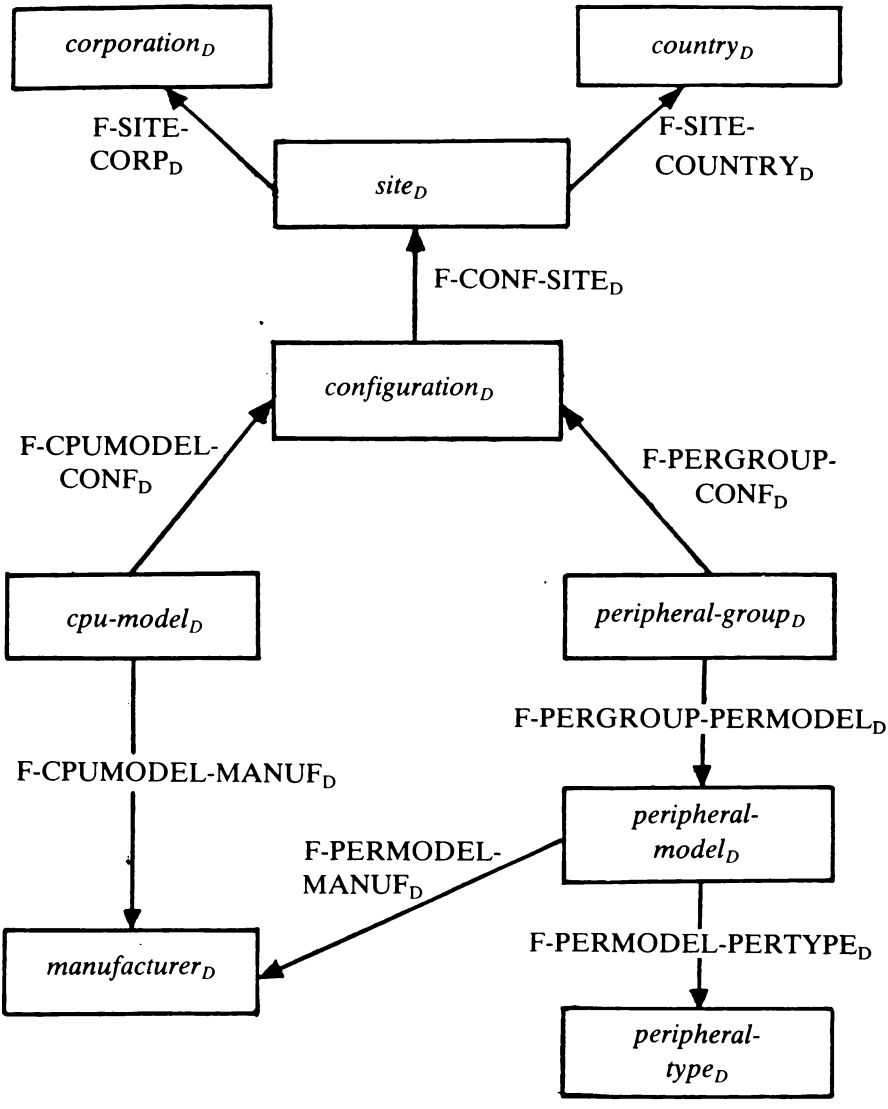

Fig. 4. DBL function constants and the types of their domains and ranges

7. **Translation.**

The relation between the languages EFL, WML and DBL is probably clear from their descriptions in the sections 4, 5 and 6 above: they are formal languages which are very similar in that they employ the same semantic operations and express them by the same syntactic structures. They differ, however, in the descriptive constants they contain. In the case of EFL these are chosen to match the descriptive words of English, whereas in the case of DBL they are chosen to match the data base primitives. WML constitutes an intermediate level which is independent of the input language as well as the data base structure. Thus, EFL, WML and DBL constitute successively "deeper" levels of analysis – they correspond to three successive steps on the path from an English question to its answer. The PHLIQA1 program may therefore be viewed as consisting of the following series of modules (see fig. 2, p. 46):

English-to-EFL translation.
EFL-to-WML translation.
WML-to-DBL translation.
DBL evaluation.

The question-answering program is thus split up into four distinct modules each carrying out a separate task. This has the important advantage that the correctness of each module can be assessed independently of the other ones.

Another advantage is, that changes in the choice of the data base, the subject domain or the input language do not affect the whole system, as can be seen by considering each of the modules:

- The English-to-EFL translation draws the semantic consequences of the syntactic structure of the sentence, the function words, and the formal aspects of the other words. The referential aspects of the English words are not analysed. Therefore, this module is independent of the subject domain (apart from the choice of the words included in the lexicon), and a possibly useful component in an otherwise quite different system.
- The EFL-to-WML translation applies rules which replace EFL constants by WML expressions, i.e. they relate words of English to the semantic primitives of the subject domain. This translation depends on the lexicon of the input language and on the subject domain, but is independent of the particular structure of the data base.
- The WML-to-DBL translation relates the subject domain primitives to data base primitives. To handle another data base about the same subject matter, only this part of the translation from English into DBL would have to be modified. Note that this module is independent of the input language; it could be exactly the same in a question-answering system for Dutch or Japanese.

More intermediate steps

The global design just discussed may be refined further. When these refinements are taken into account, the path from English question to answer consists of the following sequence of operations:

English-to-EFL translation
EFL-to-EFL$^-$ translation
Simplification
EFL$^-$-to-WML translation
Simplification
WML-to-DBL* translation
Simplification
DBL*-to-DBL translation
Evaluation
Answer-formulation.

- At most of the levels a *simplification module* is called, which converts an expression of a PHLIQA1 language into a logically equivalent, but simpler expression of the same language.
- Between the EFL level and the WML level, there is a level where the question is represented as an expression of a language called EFL$^-$. At this level, the subject domain concepts represented by English words are taken as primitive: there is one constant for every word meaning. In EFL there is one constant for every word, and in WML there is an expression for every word meaning, analysing it in terms of a limited number of primitives. Therefore EFL$^-$ is a convenient intermediate step between EFL and WML.
- Between the WML level and the DBL level, there is a level where the question is represented as an expression of DBL*. DBL* is a language which is derived from DBL: DBL* is DBL "enriched" in such a way that any WML constant can be translated into it. (See Chapter V, section 6 for details). **Not** every WML constant can be translated into DBL itself. The translation of a WML expression may "block" at the DBL* level, because certain DBL* constants cannot be translated into DBL. (See Chapter V, section 6, however, for an interesting alternative strategy which may be applied in this case.)

Each translation module performs a task which is precisely defined. For every two successive formal languages there is a set of *translation rules* which determines for any source language expression what target language expressions would be correct translations. The program accesses these

translation rules to generate, for any incoming source language expression, a corresponding target language expression.

Examples of PHLIQA1 translation rules for the EFL⁻-to-WML translation, the WML-to-DBL* translation and the DBL*-to-DBL translation are given in Bronnenberg et al. (1980). In Chapter V below, I shall give a detailed discussion of the use of translation rules for knowledge representation.

8. The Control Structure of the PHLIQA1 Program.

We have already mentioned the most important components of the PHLIQA1 program. Now we shall indicate how they cooperate to produce the over-all behaviour of the system. The flow of control in the PHLIQA1 program can best be shown by describing the algorithm in a hopefully self-explaining kind of pseudo-algol.

The description of the algorithm[6] assumes the following primitive procedures:

READ, which has no arguments, and which delivers the string last typed in by the user.

ENGLISH-TO-EFL, which takes a string as its argument, and delivers a (possibly empty) array containing its EFL translations.

EFL-TO-EFL⁻, which takes an EFL expression as its argument, and delivers a (possibly empty) array containing its EFL⁻ instances.

EFL⁻-TO-WML, which takes an EFL⁻ expression as its argument, and delivers its WML translation.

WML-TO-DBL*, which takes a WML expression as its argument, and delivers its DBL* translation.

SIMPLIFY, which takes an expression of an unambiguous PHLIQA1 language as its argument, and delivers an equivalent (simplified) expression of the same language.

EVAL, which takes a DBL expression as its argument, and delivers a value-expression which represents the answer.

ASTERISK, which takes a DBL* expression as its argument, and delivers TRUE if

[6] Compared to the actual program, this algorithm contains some simplifications. For instance:
- the actual program need not be called separately for every single question,
- it offers the possibility to correct words in the input question which do not occur in PHLIQA1's lexicon (e.g. because they are misspelled).
- the actual system contains facilities for testing it (e.g. the possibility to print out intermediate results).
- in the actual program, the testing of presuppositions is combined with the evaluation of the DBL expression (this is more efficient than the algorithm we give here).

All these and other refinements, which obscure a clear view of the most essential aspects of the algorithm, are left out here.

the expression does not belong to DBL, and FALSE if the expression does belong to DBL.
PRESUP-EXPRES takes an expression as its argument, and delivers the presupposition-expressions of the expression.[7]
ANSWER takes a value-expression as its argument, and delivers a string.
PRINT takes a string as its argument, and displays it on the terminal.
The program uses the following variables:
X_1, X_2, X_3, X_5 (whose values are PHLIQA1 expressions),
buftext (whose values are strings),
buflevel (whose values are integers), and
false-presups (whose values are truthvalues).
In terms of the above procedures and variables, a simplified version of the PHLIQA1 program can be specified as follows:

```
begin
buflevel := 0; buftext := "your question is considered ungrammatical";
for X1 through ENGLISH-TO-EFL (READ) do
(if buflevel < 1 then (buflevel := 1;
                       buftext := "your question is meaningless in the subject
                                  domain");
for X2 through EFL-TO-EFL⁻ (X1)
do (
X3: = SIMPLIFY (WML-TO-DBL* (SIMPLIFY (EFL⁻-TO-WML (SIMPLIFY(X2)))));
if ASTERISK (X3)
then (if buflevel < 2
      then (buflevel := 2;
            buftext: = "the database does not contain the information which
                       is needed to answer your question"))
else (false-presups := FALSE;
      for X5 through PRESUP-EXPRES (X3) do
      (if EVAL (X5) = FALSE then false-presups := TRUE);
      if false-presups
      then (if buflevel < 3 then (buflevel := 3;
                buftext := "your question contains a false presupposition"))
      else (PRINT(ANSWER(EVAL(X3)));
            buflevel := 4; buftext := "  ";
            PRINT ("Do you want a search for another interpretation of your
                   question?");
                  if READ = "no" then exit)) ));
PRINT (buftext)
end
```

[7] See Bronnenberg et al. (1980), section 5.5, for the representation of presuppositions in PHLIQA1-expressions.

9. The Behaviour of the PHLIQA1 System.

A translation module does not necessarily translate an incoming expression into one expression at the next lower level: the expression may also have no translation at all (the expression is "blocked"), or more than one translation (it is "ambiguous").

The system can provide all answers to an ambiguous question, but because it does not contain a module which translates from, for instance, WML into English, it cannot indicate to the user which answer belongs to which interpretation of the question. Two kinds of ambiguity may occur:

- *Syntactic ambiguities*, which arise during the translation from English into EFL, when a question can be parsed in more than one way. An example is the question "What companies have a configuration with a cpu that costs more than 100.000 dollars?", where the relative clause "that costs more than 100.000 dollars" can be combined with the nominal phrase "cpu" or with the nominal phrase "configuration with a cpu".

- *Semantic ambiguities,* which arise during the translation from EFL into EFL^-, when an EFL term (corresponding to an English word) can be translated in more than one way into EFL^-. For instance, the EFL predicate "have" can either mean "have-as-part" (as in "Does Akzo's configuration have two card readers?") or "possess" (As in: "Does Akzo have two card readers?").

When there is an ambiguity, the system investigates one of the possibilities, until it either blocks at a certain level or leads to an answer. After an answer has been given, the user is asked whether he wants the system to investigate other analyses of the input question. If he does, and also in the case of blocking, the system backs up to the last point where there was an alternative possibility. It then starts working on this alternative – and so on, until the user is no longer interested in other analyses or until no other analyses are found any more.

If there is no analysis that leads to an answer, the system considers the analysis that reached the "lowest" level to be the most plausible one, and a message indicating the reason for the blocking at that level is presented to the user. The possibility of using this mechanism (described more precisely in section 8 above) is one of the beneficial consequences of the distinction between different semantic levels.

Let us now take a closer look at the reasons why blocking may occur at the different levels.

- A question may contain words which do not occur in the lexicon that is used for the English-to-EFL translation. In such a case the system indicates these words; the user of the system can then substitute other words for them, or reformulate the question altogether.
- It may be impossible to translate a question into EFL, although it only contains "legitimate" words: the system considers it to be "ungrammatical". This happens when a haphazard sequence of words is typed in ("The of computers are what?"), or when there is a departure from standard English syntax ("What computers is there in Germany?"); but a sentence may also be rejected because the syntax of the system is more limited than we would like it to be. ("What companies possess more computers than Shell?" is rejected because the syntax has no rules for elliptic comparative clauses). In all these cases, the system states that it considers the sentence to be ungrammatical.
- An EFL analysis of a question may be not translatable into EFL^-. This means that it may make sense in some context, but not in the context of PHLIQA1's subject domain. This would occur if the expression contained a constant corresponding to an English word that had no meaning in the context of this specific subject domain. That does not happen in practice, however, because such words were not put in the dictionary. As a more interesting example, let us consider the question "What is the price of Germany?" Although the notion "price" as well as the notion "Germany" are represented in the EFL^- language, the EFL analysis of this question does not lead to a semantically well-formed EFL^- expression, because the EFL^- function "price" is not applicable to the elements of the EFL^--set "countries"; in the subject domain of PHLIQA1, countries don't have a price. If no EFL analysis of a question leads to a semantically well-formed EFL^- expression, the reason for this is reported to the user of the system.
- A DBL* analysis of a question may be not translatable into DBL, because DBL lacks the constants that would be needed to represent it. This means that certain information is consistently lacking in the data base. For example, let's consider the question "When were Akzo's card readers installed?" The data base contains the installation dates of configurations, i.e. of the cpu with the peripherals it has then; other peripherals may have been added later, but their installation dates are not stored, which means that the installation dates of individual card readers are not known. The system considers questions about them as meaningful however, so they get a WML analysis, but the factual knowledge to answer them is lacking. Therefore, a message is displayed which indicates why the translation into DBL did not succeed.
- A question may get a DBL analysis that denotes an answer which is in fact not the most appropriate response, because the question made undue

presuppositions about the actual state of affairs in the subject domain. The system checks the presuppositions that a question makes. As an example we may consider the question "How expensive is the computer owned by Shell?" The system takes this as presupposing that Shell owns exactly one computer.[8] If such a presupposition fails to hold, the system points that out instead of giving an answer.

Summarizing, we may say that the multilevel structure of the PHLIQA1 system offers many possibilities for flexible and cooperative system replies to user queries. Some of these possibilities have already been implemented in the existing PHLIQA1 system, but many others remain to be explored in the future.

The next chapters are devoted to a discussion of the theoretical issues involved in some important design decisions that PHLIQA1 was built on: the assessment in terms of logical model theory of the way in which an "ordinary" data base may be said to represent knowledge (Chapter IV), and the development of knowledge representation by means of translation rules, in order to bridge the "semantic gap" between English and the data base (Chapter V).

[8] Since every question is treated separately, the system ignores the possibility that the class of computers that should be taken into consideration is constrained by the foregoing part of the discourse.

Chapter IV. Data Bases as Value Specifications.

1. **Introduction.**

In real-world, non-experimental computer programs which answer queries about the state of affairs in a subject domain, the state of affairs is normally represented by a *data base* – a formatted collection of data stored on magnetic disk or in another form of mass memory.

Many question answering systems (including PHLIQA1) are, in fact, "natural-language data base interfaces" which are meant to function as the front end of a query evaluation system which employs an existing data base management system. Therefore, a satisfying account of operations on data bases is an important component of a theory of computational question answering.

In the treatment of questions and answers developed in Chapter II, both questions and answers were represented as expressions in a logical language and the connection between such expressions and states of affairs in the subject domain was constructed through logical model theory. To enable an account of the way in which a data base may be used to answer questions, the meaning of a data base must be described in terms of the same basic semantic notions that were used earlier to account for the meaning of questions and answers. We must therefore establish a connection between the content of a data base and the possible interpretations of a logical language.

The main thesis of this chapter is that this connection is, in fact, quite simple. We shall argue that any data base schema should be considered as specifying the descriptive constants of a logical language, while any particular data base within a schema specifies an interpretation of that logical language. The notion of a "value specification" – a formal object which induces an interpretation on a logical query language – will figure prominently in the discussion. This perspective is being advanced as an alternative to the rather widespread idea that data bases should be analyzed as collections of first order axioms. (See section 6, below). An important argument in its favour is that it accounts in a very direct way for the correctness of recursive query evaluation procedures.

The viability of the perspective on data bases advocated here has been pointed out before. (Scha, 1977; Nicolas and Gallaire, 1978; Bronnenberg et. al., 1980; Konolige, 1981. Important connections with Codd's work (1970) will be considered specifically in section 4 below). However, no comprehensive formal articulation of this way of conceptualizing data bases has been presented previously.

2. Value Specifications.

We have argued earlier that a subject domain may be characterized by a definition of the descriptive constants of a logical language and their intuitive meanings. The state of affairs in a subject domain could then be characterized by an *interpretation* of this language, which specifies the denotation of every descriptive constant. In the present chapter, we shall show that specifying a formal object which indicates the denotation of every constant in the query language – i.e. the extension of the interpretion function – is a way of specifying the state of affairs in the subject domain which ties in directly with the definition of the semantics of the query language.

To be able to construct such a formal object, we assume that there is a "value language" corresponding to the query language, which has a "logical proper name" for every individual in the domain. This language has exactly the same syntactic constructions and the same semantics as the query language as well as exactly the same type system, but it contains no descriptive constants. The value language is syntactically defined as a logical language with only "individual" constants (i.e. the type of every constant is an atomic type). The semantics of a value language is defined in the usual way with two extra constraints on the interpretations of the language[1]:

1. Every individual constant denotes a distinct entity
2. Every entity in the domain of a descriptive atomic type is denoted by an individual constant.

A *value specification* may now be defined as a collection of pairs $\langle c,e \rangle$ which specifies the extension of a function which assigns to every descriptive constant c of the logical language an expression e of the value language. Since the value language expressions have the same denotation for every interpretation of that language, this function induces the one interpretation

[1] This way of adapting the definition of interpretations is not as ad hoc or far-fetched as it may seem. Logical systems in which individual constants were given this special role have a venerable history (e.g. Wittgenstein (1922), Carnap (1947) and Reiter (1977)). A difficulty with this approach is that the language fixes the number of individuals in the domain, although for data bases about a fragment of the real world, the individuals in the domain are not usually fixed. (Pott, 1976). There are at least three alternative ways of dealing with this problem:

1. Adopt a "loose ontology" by assuming a fixed, infinitely large domain of *possible individuals*. The set of *actual individuals* is then defined as the union of the domains of the atomic types, constituting a subset of the possible individuals.
2. Abstain from defining one particular value language. Instead define the set of all possible value-languages where these languages differ from each other in having different sets of individual constants.
3. Formulate the queries so that they do not refer to real world entities. (Questions which ask about real world entities may be transformed into questions which ask about their names and other "formal" properties (Chapter II, section 7). The technique of "identification-translations" may be used to identify real world entities in terms of their names (Chapter V, section 4).)

on the logical language which assigns to every descriptive constant the denotation of the expression assigned to it by the value specification.

There is an extremely simple algorithm which, given a value-specification, could be used to turn any query expression into an expression which satisfies many of the requirements of an adequate answer-expression. Simply by substituting values for constants according to the value specification, an L-determinate expression is generated which has the same denotation as the query expression. However, this expression is, in general, unnecessarily complicated. Instead, a recursive evaluation algorithm is therefore used to turn a query expression into a *canonical value expression* which can be used as an answer.[2] Exactly how such an algorithm functions will be discussed after describing the nature of canonical value-expressions.

Canonical value-expressions

For every finite denotation of the value language expressions, we define one *canonical value expression* with that denotation.

If a data base implements a value-specification, a query-expression may be evaluated by means of a collection of recursive procedures which correspond closely to the recursive definition of the semantics of the language. These procedures operate on canonical representations of the values of various types.

The canonical representations of values may be defined as the expressions of a *canonical value-language*:

1. Every individual constant is a canonical value-expression.
2. If $A_1, \ldots, A_n$ are canonical value-expressions of types $\alpha_1, \ldots, \alpha_n$, then:
   set($\langle A_1, \ldots, A_n \rangle$) is a canonical value-expression of type $S(\cup(\alpha_1, \ldots, \alpha_n))$,
   bag($\langle A_1, \ldots, A_n \rangle$) is a canonical value-expression of type $B(\cup(\alpha_1, \ldots, \alpha_n))$,
   list($\langle A_1, \ldots, A_n \rangle$) is a canonical value-expression of type $L(\cup(\alpha_1, \ldots, \alpha_n))$,
   file($\langle A_1, \ldots, A_n \rangle$) is a canonical value-expression of type $F(\cup(\alpha_1, \ldots, \alpha_n))$.
3. If $A_1, \ldots, A_n$ are canonical value-expressions of type $\alpha_1, \ldots \alpha_n$, then $\langle A_1, \ldots, A_n \rangle$ is a canonical value-expression of type $\langle \alpha_1, \ldots, \alpha_n \rangle$
4. If $\langle A_1, B_1 \rangle, \ldots, \langle A_n, B_n \rangle$ are value-expressions of type $\langle \alpha_1, \beta_1 \rangle, \ldots, \langle \alpha_n, \beta_n \rangle$, resp., then **function** $(\langle\langle A_1, B_1 \rangle, \ldots, \langle A_n, B_n \rangle\rangle)$ is a value-expression of type $(\ (\alpha_1, \ldots, \alpha_n) \rightarrow\ (\beta_1, \ldots, \beta_n))$
5. If N is a canonical value-expression of type *integer* or *real* and E is a canonical value-expression of type ε, then *(num: N, unit: E)* is a canonical value-expression of type AMT (ε).

[2] This is also the case in PHLIQA1. In Bronnenberg et al. (1980) the PHLIQA1 program was described misleadingly as using the substitution algorithm.

6. If A is a canonical value-expression of type α then, for any integer i, $\mathbf{id}_i\ (A)$ is a canonical value-expression of type $\mathrm{ID}_i\ (\alpha)$.

A *normalized value specification* assigns to every descriptive constant of a logical language a canonical expression of the corresponding value language. Canonical values may be used, therefore, to formalize the definition of the semantics of a logical language by correlating every syntactic rule which forms an expression out of sub-expressions with a semantic rule which defines the denotation of the expression in terms of the denotations of the sub-expressions.[3] Together, these semantic rules define the denotation of any expression in terms of the denotation of the constants occurring in it. If canonical values have been defined, we may define formal correlates of the semantic rules for the case of finite denotations as functions which define the value of an expression in terms of the value of its sub-expressions.[4] These functions, taken together, define the value of any expression in terms of the values of its constants.

Given a formalization of the semantic definition of a logical language, we may construct an algorithmic definition. Every function which defines the value of a certain kind of expression in terms of the values of its sub-expressions, may be implemented by an effective procedure which computes the value of the expression on the basis of the values of its subexpressions. Together, these procedures constitute an effective recursive algorithm for computing the value of an expression on the basis of the values of the constants occurring in it. Thus, a value specification of the normalized variety makes it possible to compute the answer to a query which is formulated in the appropriate logical language, by means of a simple recursive evaluation procedure.

More refined versions can be easily imagined. For instance, the PHLIQA1 procedures which compute the values of quantification-, selection- and iteration-expressions do not always evaluate both their sub-expressions. They have two sub-expressions, one denoting a set, the other denoting a function. If the function-expression is a lambda-expression, its whole extension is not computed. Instead, the value of the set-expression is used as the range of the lambda-variable, and only the values of the relevant instances of the body of the lambda-expression are computed.

It is clear that value-specifications may be implemented as data bases of some sort. The descriptive constants of the logical language may then be

[3] For the sake of simplicity of expression, the notion "denotation of sub-expressions" is taken to encompass the domains of the types of (binding occurrences of) variables.

[4] This means that we must restrict our attention to expressions not containing formal constants with infinite denotations. Variables ranging over infinite formal domains are thus also excluded.

viewed as defining data base schemata. All value-specifications for these constants may be stored as homogeneous collections of data according to a format which is determined by the type of the constant. Conversely, the most important data models underlying existing data base management systems can be seen as implementations of value-specifications. In the next sections, we shall show how this is the case in the Relational Model and the CODASYL Model, and argue that other "abstract data models" can be formalized in this way as well.

4. **Relational Data Bases viewed as Value Specifications.**

Codd (1970) defined *relational data bases* as collections of tables, where a table is a set of n-tuples of individuals. The elements within every n-tuple may be identified by the integers $1....n$ or by n names called *attributes*. An n-tuple within a table is identified by the *keys* of the table, a subset of the attributes. The relational terminology and query languages like Relational Algebra suggest that data bases be viewed as specifying the values of constants denoting sets of n-tuples. The fact that certain elements identify the n-tuple in which they occur, however, leads to a different perspective.

Instead of saying that a table specifies the extension of an n-ary relation, we say that it specifies the extension of a partial function from k-tuples to n–k-tuples, if k elements of a tuple constitute the primary key together. This means that a table is read as a translation rule of the form

$$F \implies \textbf{function}\,(\langle \textbf{tuple}_2\,(\textbf{tuple}_k\,(B_{11}, \ldots, B_{1\,k}), \textbf{tuple}_{n\text{-}k}\,(B_{1k+1}, \ldots, B_{1n})),$$
$$\textbf{tupple}_2\,(\textbf{tuple}_k\,(B_{m1}, \ldots, B_{mk}), \textbf{tuple}_{n\text{-}k}\,(B_{m\,k+1}, \ldots, B_{mn}))\rangle)$$

where F is a query-language constant of type
$(\langle \alpha_1, \ldots, \alpha_k \rangle \rightarrow \langle \alpha_{k+1}, \ldots, \alpha_n \rangle)$

The types $\alpha_1, \ldots, \alpha_n$ are, in relational terminology, the domains of the relation A: they indicate the set of values that may occur in the corresponding "slot" in the n-tuple. This domain must either be specified in the data base (by occurring as the domain of a single key of a relation) or it must be known independently of the data base (in our terminology: it is a formal type, such as *integer* or *string*).

Consider, for example, the relation S in Date (1975). This table represents information about suppliers, identified by a code which is the value of the attribute S#. The table gives the name, the status and the city of every supplier:

S

S#	NAME	STATUS	CITY
S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	30	Paris
S4	Clark	20	London
S5	Adams	30	Athens

The table can be read as the translation rule (2)

$F_S \Longrightarrow$ function (<
<<S1>, <"Smith", 20, "London">>
<<S2>, <"Jones", 10, "Paris">>
<<S3>, <"Blake", 30, "Paris">>
<<S4>, <"Clark", 20, "London">>
<<S5>, <"Adams", 30, "Athens">>>) (3)

We are arguing that this view of relational data bases has advantages over the more strictly relational view which would consider a table in a data base as a translation rule of the form

$$A \Longrightarrow \textbf{set}\,(\langle \text{tuple}_n\,(B_{11}, \ldots, B_{1n}), \text{tuple}_n\,(B_{m1}, \ldots, B_{mn})\rangle), \quad (4)$$

where A is a query-language constant of type S ($\langle\alpha_1, \ldots, \alpha_n\rangle$).
The relation S above would then be analyzed as

S $\Longrightarrow$ **set** (<tuple_4 (S1, "Smith", 20, "London"),
tuple_4 (S2, "Jones", 10, "Paris"),
tuple_4 (S3, "Blake", 30, "Paris"),
tuple_4 (S4, "Clark", 20, "London"),
tuple_4 (S5, "Adams", 30, "Athens")>) (5)

If the data base is analyzed as (5), there would be a problem justifying that a formula such as

$$\text{tuple}_4\,(\text{"S1"}, \text{"Jones"}, 30, \text{"London"}) \in S \quad (6)$$

can be evaluated as FALSE on the basis of (2) by only inspecting the first entry. What would be needed to account for this would be an axiom like

$$\forall u, x_1, x_2, y_1, y_2, z_1, z_2: ((\mathbf{tuple}_4\,(u, x_1, y_1, z_1) \in \mathrm{S}\ \&\ \mathbf{tuple}_4\,(u, x_2, y_2, z_2) \in \mathrm{S}) \supset (x_1 = x_2\ \&\ y_1 = y_2\ \&\ z_1 = z_2))$$

Evaluation procedures actually used in data bases, however, do not make use of such axioms. Their operations can be accounted for by an analysis such as (3). In terms of that perspective on data bases, (6) would be formulated as

$$\mathrm{F_S}\,(\langle \mathrm{S_1} \rangle) \quad = \quad \langle \text{"Jones"}, 30, \text{"London"} \rangle \tag{7}$$

By consulting value-specification (3), this can be equated to

$$\langle \text{"Smith"}, 20, \text{"London"} \rangle \quad = \quad \langle \text{"Jones"}, 30, \text{"London"} \rangle \tag{8}$$

and further to

$$\text{FALSE} \tag{9}$$

As another example of the data base analysis we propose, consider the relation SP from Date (1975), represented by the following table:

SP

S#	P#	QTY
S1	P1	3
S1	P2	2
S1	P3	4
S1	P4	2
S1	P5	1
S1	P6	1
S2	P1	3
S2	P2	4
S3	P3	4
S3	P5	2
S4	P2	2
S4	P4	3
S4	P5	4
S5	P5	5

(10)

The *combination* of the values of the attributes S# (supplier-identification) and P# (part-identification) identifies a row in this table. The table can be read as representing the following rule:

F_{SP} $\Longrightarrow$ **function** (<
<<S1, P2>, <3>>,
<<S1, P2>, <2>>,
<<S1, P3>, <4>>,
<<S1, P4>, <2>>,
<<S1, P5>, <1>>,
<<S1, P6>, <1>>,
<<S2, P1>, <3>>,
<<S2, P2>, <4>>,
<<S3, P3>, <4>>,
<<S3, P5>, <2>>,
<<S4, P2>, <2>>,
<<S4, P4>, <3>>,
<<S4, P5>, <4>>,
<<S5, P5>, <5>>>) (11)

5. CODASYL Data Bases viewed as Value Specifications.

A CODASYL system (CODASYL DBTG, 1971) can be used to implement many different kinds of abstract data models (e.g. relational data bases – see Lacroix (1977)). But a CODASYL data description may also be considered as an abstract data model in its own right. It is sufficiently well-structured for that, and the efficiency of an implementation may be enhanced if there is a direct correspondence between the query language and the data base as implemented, without intermediate models. We shall now discuss how the constants of a query-language may be derived from a CODASYL data base which only uses the most important CODASYL concepts.

Described in CODASYL terminology, a CODASYL data base is a specification of the extensions of record types, attributes and link-sets.[5] This can easily be translated into standard mathematical terminology.

A *record type* is a set of individuals.

An *attribute* is a function which has one of the record types as its domain of application, and which has as its range a subset of the strings or the integers. The record types and attributes of CODASYL may be treated just as the relations and their "slots" in a relational data base as discussed above with the difference that CODASYL allows record types without identifying external keys. In that case the *data base keys* which are used to identify the records of that type must be constants of the value language; but they need not be part of the query-language and they cannot be mentioned in queries although they can occur as values of query expressions.

[5] For the sake of simplicity we ignore some less central CODASYL concepts: the possibility of using "aggregate attributes" and of sorting the records of a given record-type.

A *link-set* (or, in CODASYL terminology, simply and misleadingly called a *set*) represents a function which has the individuals corresponding to a record type as its range and the individuals corresponding to another record type as its domain. A CODASYL data base management system stores the extension of such a function F in such a way that not only the result of the application of F to an argument is more or less directly available, but also the result of the application of its inverse F^{-1} to an argument.

In the PHLIQA1 data base, for instance, the link set COUNTRY-SITES specifies a function F-SITE-COUNTRY, from site records to country records and its inverse, F-SITE-COUNTRY^{-1}, from country records to sets of site records. For each site record S there is an occurrence of the link-set COUNTRY-SITES which has S as a MEMBER and the country record G which is the OWNER of this link-set occurrence is the value of the function F-SITE-COUNTRY for the argument S.[6)]

It is clear that, given a CODASYL data base, the descriptive atomic types and the descriptive constants of a corresponding Data Base Language can be derived in a systematic way: for every record type R we have a descriptive atomic type α_R[7)] and for every attribute of record type R, we have a function constant of type $(\alpha_R \rightarrow \textit{string})$ or $(\alpha_R \rightarrow \textit{integer})$, depending on the kind of values of the attribute. For every link-set that has R as its OWNER record type and S as its MEMBER record type, we have a function of type $(\alpha_S \rightarrow \alpha_R)$, and its inverse, a function of type $(\alpha_R \rightarrow S(\alpha_S))$.

An illustration of this way of deriving the constants of a logical query-language from a CODASYL specification of a data base can be found in Chapter III, section 6, where this method is applied to the PHLIQA1 data base.

Conclusion

Our discussion of the Relational Model and CODASYL indicates that the idea of viewing data bases as value specifications has considerable generality. It is not limited in any way to one particular data model – instead, it provides a general method for analysing data bases which is equally applicable to all well-defined data models. The differences between different data models simply appear as differences in the structure of the semantic types of the descriptive constants in the corresponding logical languages. The *Functional Dependency Model*[8)], to mention one other example, yields to the same

[6)] Similarly, for any given country record G, there is a link-set occurrence which has G as its OWNER. The set of MEMBERS of this link-set occurrence is the value of F-SITE-COUNTRY^{-1} for the argument G.

[7)] And, therefore, the corresponding constant $GS\alpha_R$ denoting the domain of α_R.

[8)] See Housel et al., 1979.

treatment without difficulty, while leading to different types of function constants than the models discussed above.

Note that in all these cases the value specification analysis of a data base captures the completeness of the files of records, and the fact that one entity has only one value for a given attribute[9]. This is an important advantage compared to the "axiom set" analysis of data bases, that we discuss in the next section.

5. **Data Bases as Axiom Sets.**

When notations of formal logic are brought to bear on data base matters, data bases are usually analysed as sets of first-order axioms.[10] We shall now briefly discuss why we have not chosen this alternative.

The axiomatic analysis is usually applied to relational data bases. A relational data base P, consisting of n-tuples of the form $\langle x_1, \ldots, x_n \rangle$ is then analysed as a set of ground literals of the form $P(x_1, \ldots, x_n)$. For instance, the relation S discussed in section 4 above would be seen as specifying the axioms:

$S(S_1, \text{"Smith"}, 20, \text{"London"})$,
$S(S_2, \text{"Jones"}, 10, \text{"Paris"})$,
$S(S_3, \text{"Blake"}, 30, \text{"Paris"})$,
$S(S_4, \text{"Clark"}, 20, \text{"London"}$,
$S(S_5, \text{"Adams"}, 30, \text{"Athens"})$

The direct connection with recursive evaluation procedures, which exists in the case of value-specifications, is lost when a data is viewed as an axiom set. An axiom set, as a syntactic/semantic object, stipulates the truth of individual formulas. A first-order axiom set does not make the extensions of predicates or functions directly available to a query evaluation algorithm.

Some additional complications are worth mentioning. If a file is assumed to be complete – a common enough case – a set of axioms having the form of *negative* literals must be assumed to be specified implicitly by the data base. This may be done in different ways. One may assume a simple kind of abbrevation mechnism: every literal which does not occur in the data base is known to be false.[11] One may also adopt the more complex "Closed World

[9] Elegant techniques for dealing with *incompleteness* exist as complements to the value specification analysis. See Chapter V, section 6.

[10] See, for instance, the papers in Gallaire and Minker (1978).

Assumption" (Reiter, 1978a; see Chapter VI, section 5 below). In both cases, the data base does not present the axiom set it stands for in a direct way. The axiom set which describes the data base content is a "virtual" object – it is not the actual syntactic object that the query evaluation procedure interacts with.

As a last point we may mention that within the axiomatic approach one often uses first-order logic without function constants. As we discussed in section 3 above, this leaves important properties of a data base unaccounted for.[12)]

Summarizing, we find that the properties of a relational data base are accounted for in a more satisfying way in the value specification approach than in the axiom set approach. On top of that, the value specification approach is trivially generalizable to other data models, whereas the situation with axiom sets is less clear in this respect. We conclude that value specifications are not only a possible alternative to the usual axiom set analysis of data bases, but that this alternative is in fact the preferable one.

[11)] See Wittgenstein's (1922) notion of a "picture of the world". Carnap's (1947) notion of a "state description" requires a specification of positive and negative literals. If the individuals of the universe are fixed in advance, a state description can be indicated by only specifying the positive literals. (See Biller and Neuhold, 1978.)

[12)] See Stenius (1960), for an interesting attempt at refining Wittgenstein's notion of a "picture of the world" without introducing function-constants in the logical language. The reader may compare his proposal with our alternative analysis in section 3 above.

Chapter V. Translation Specifications: a Technique for Representing the Conceptual Information of a Question Answering System

1. Conceptual Information: the Bridge between Different Levels of Meaning Representation.

As I showed in the previous chapter, a data base represents information concerning the state of the world, by specifying the extensions of concepts. To be able to use a data base to answer natural language questions, a question answering system must also possess a different kind of knowledge concerning the concepts involved: knowledge about the relation between the "data base concepts" and the concepts which the natural input-language provides for talking about the subject of the data base. Various methods may be employed for representing and using such knowledge concerning the relations between concepts.

In the design of a question answering program, how to incorporate this "conceptual information" in the system is an important decision. In this decision, one must try to strike an optimal balance between the generality of the knowledge representation formalism employed, and the effectiveness of the procedures by means of which the knowledge so represented is brought to bear on the questions the system tries to answer.

In the present dissertation, I want to discuss those methods for representing conceptual knowledge that I find reasonably well-defined. The present chapter introduces the method of "translation specifications" which I developed jointly with Jan Landsbergen, and which was implemented in the PHLIQA1 system. In the next chapter, I discuss other techniques with an equally firm model-theoretic basis.

As described in detail in Chapter III, the PHLIQA1 system uses a sequence of levels. At each level a different logical language is used to represent the content of a question. The system translates the content of an incoming query from one level to another in succession until it finally reaches the data base level where it forms the input to a component which actually computes the answer.

The present chapter deals with the knowledge representation method underlying the PHLIQA1 algorithm although the details of the algorithm itself are not dealt with. In illustrating the use of the knowledge representation method, the intermediate levels used in the system will be ignored; only two levels of representation will be assumed. An English-oriented Formal Language EFL′ which corresponds closely to the English

formulation for questions is the highest of these levels. A Data Base Language DBL which corresponds closely to a relational data base is used at the lower level.

An English-oriented level of meaning representation.

At the highest level of meaning representation that we want to consider, the meaning of a question is represented by an expression of a logical language (i.e. a formal language with an unambiguously defined model-theoretic semantics) in a way which is as close as possible to the semantic structure of the English formulation, and as independent as possible of specific features of the subject domain that the question refers to. The logical language EFL′ that may be used for this purpose, is a language which contains a descriptive constant for every descriptive lexical item of the input language. Since distinctions between different kinds of objects in the subject-domain cannot be made independently of the subject-domain, there is only one descriptive atomic type at this level, which I will call *entity*.[1] Therefore, the semantic types of the descriptive constants may be systematically related to the syntactic categories of the corresponding lexical items (as in Montague (1973)).

For example, in the illustrations I shall use in the next sections of this chapter, I shall assume for every noun a constant with type S(*entity*), denoting the set of individuals which fall under the description of this noun: corresponding to "employee" and "employees" there is a constant EMPLOYEES denoting the set of all employees. Corresponding to an n-place verb there is an n-place predicate, i.e. a constant of type (<*entity*, ..., *entity*> ⟶ *truthvalue*). For instance, the verb "have" corresponds to the 2-place predicate HAVE – a constant of type (<*entity*, *entity*> ⟶ *truthvalue*).

Thus, the input analysis component of a question answering system may translate the question.

"How many departments have more than 100 employees?" (1)

into

$$\textbf{Count}\,(\{x \in \text{DEPARTMENTS} \mid \textbf{Count}\,(\{y \in \text{EMPLOYEES} \mid \text{HAVE}\,(x,y)\}) > 100\}) \qquad (2)$$

where both x and y have the type *entity*.[2]

[1] Thus, EFL′ shares features with EFL and with EFL⁻, as they are defined in Chapter III. Like EFL⁻, EFL′ has unambiguous constants. But like EFL, it only has one descriptive atomic type.

[2] If the input analysis component operates in a compositional fashion, i.e. constructs the meaning of the sentence by means of a recursive procedure which constructs the meaning of a constituent out of the meanings of its subconstituents, this component would not produce (2) directly. Instead, it would produce a more complicated equivalent expression, which could be automatically simplified by a procedure performing simple equivalence-transformations such as β-reduction. The output of that procedure could then be an expression like (2) above.

A data base oriented level of meaning representation.

A data base specifies an interpretation of a logical language by specifying the values of its descriptive constants. What these constants are, follows directly from the structure of the data base. To illustrate this, we shall use a simple relational data base. (In Chapter III, in discussing PHLIQA1, a CODASYL data base was used as an example.) Assume that a data base has a file with records which represent the employees of a firm, and that this file has an attribute indicating the department of each employee. A file is also present with records which represent the departments of the firm. This part of the data base may be viewed as specifying the extension of a constant EMPS of type S(*entity*) standing for the set of employees,[3] of a constant DEPTS of type S(*entity*) standing for all departments and of a function F-EMP-DEPT of type (*entity* ⟶ *entity*) assigning to every employee a department.
In terms of such a data base structure, question (1) may be formulated as (3):

How many departments have more than 100 employees? (1)

$$\textbf{Count}\,(\{x \in \text{DEPTS} \mid \textbf{Count}\,(\{y \in \text{EMPS} \mid \text{F-EMP-DEPT}(y) = x\}) > 100\}) \quad (3)$$

Thus, we have two alternative formulations for the query – formulation (2), in terms of an "English-oriented Formal Language" EFL′, and formulation (3), in terms of a "Data Base Language" DBL.

The connection between EFL′ and DBL.

EFL′ and DBL are two different logical languages with different constants which can nevertheless "talk about the same subject domain". A question is represented initially as an EFL′ expression, while the state of affairs in the world is specified by a DBL interpretation. The conceptual information of the system represents the connection between EFL′ and DBL. Given a DBL interpretation, it defines the set of EFL′ interpretations compatible with it. The set of possible answers to an EFL′ query, therefore, is the set of possible answers allowed by the conceptual information given the state of the world as represented by the data base.

There are different kinds of possible relationships between EFL′ and DBL, which may vary in their degree of complexity. The discussion of this chapter begins with one simple but important case. This will provide a

[3] This analysis of a relational data base is somewhat simpler than the one proposed in Chapter III where a many-sorted type system was used. The many-sortedness will be reintroduced in section 3 of the present chapter.

starting point for the gradual development of more sophisticated methods later on.

2. Conceptual Information in the Form of Translation Rules.

2.1. Translation Rules.

Consider the situation that all the concepts represented by the constants of EFL′ can also be represented by expressions of DBL. In this case, the relation between the two languages can be described by specifying for any EFL′ constant a "synonymous" DBL expression. Thus, the conceptual knowledge of the system is represented as a *constant-translation*: the specification of the extension of a function CT which maps the descriptive constants of a source language (i.e. EFL′) into the expressions of a target language (i.e. DBL). The intention is that CT defines constant c as being equivalent to expression CT(c) in all interpretations of source language and target language which are compatible with each other. Or, put slightly differently, given an interpretation of the target language, a constant-translation defines at most one interpretation of the source language as being compatible with it: the interpretation which assigns to every descriptive constant c the denotation of CT(c). (To formal constants it assigns the same entities as the target language interpretation.)

If the type systems of source language and target language are identical, the translation algorithm complementing this method of knowledge representation is trivial: it inspects the source language expression and substitutes for every constant c the target language expression CT(c) that defines it. As Leibniz (1686) put it: "... it does not seem to me that there is need for any other kind of proof than one which depends on the subsitution of equivalents" (cf. Ishiguro, 1977, p. 17).

As an example application of this method, consider the example data base of the previous section. The data base language considered has the descriptive constants EMPS, with type S(*entity*), DEPTS, with type S(*entity*), and F-EMP-DEPT, with type (*entity* $\dashrightarrow$ *entity*).
The EFL-to-DBL translation is now defined by the rules

DEPARTMENTS $\Longrightarrow$ DEPTS
EMPLOYEES $\Longrightarrow$ EMPS
HAVE $\Longrightarrow$ $(\lambda u, v:$ F-EMP-DEPT$(v) = u)$

These rules can be directly applied to the formula (2) which represents question (1). Substitution of the right hand expressions for the left hand constants in (2) yields (4), which is equivalent to (3) above.

How many departments have more than 100 employees? (1)

$$\textbf{Count}(\{x \in \text{DEPARTMENTS} \mid \textbf{Count}(\{y \in \text{EMPLOYEES} \mid \text{HAVE}(x,y)\}) > 100\}) \quad (2)$$

$$\textbf{Count}(\{x \in \text{DEPTS} \mid \textbf{Count}(\{y \in \text{EMPS} \mid (\textit{fun:}\ (\lambda u,v:\ \text{F-EMP-DEPT}(v) = u),\ \textit{arg:}\ \langle x,y\rangle)\}) > 100\}) \quad (4)$$

Notice that this method requires a certain "richness" of the logical language. If the language does not have λ-abstraction, one can not say, for example

$$\text{HAVE} \Longrightarrow (\lambda u,v:\ \text{F-EMP-DEPT}(v) = u).$$

Instead, one would have had to say something like

$$\text{HAVE}(u,v) \Longrightarrow \text{F-EMP-DEPT}(v) = u.$$

This means that instead of a *local* substitution rule there would be a schema of *global* rules, Thus, an inherently more complex framework would be employed, which brings its own problems.[4)]

2.2. **Type constraints.**

The definition of the logical language used in the PHLIQA1 system (see Appendix A) does not allow arbitrary combinations of constants, variables and formal operators as language expressions. Since many combinations would in fact be meaningless, the language definition explicitly constrains the possible combinations of elements to the meaningful ones. The type system which accomplishes this task may be used in a similar way to constrain the allowable translation rules.

In order to guarantee that the result of the translation procedure indicated in the previous subsection is a semantically well-formed expression of the target language, as defined in Appendix A, section 5, we impose conditions on the constant-translation concerning the semantic type of the source language constants and the semantic types of their target language translations. We require, therefore, that for any constant c,

$$\text{TYPE}(\text{CT}(c)) \overset{\subseteq}{\mathbf{T}} \text{TYPE}(c).$$

The relation $\overset{\subseteq}{\mathbf{T}}$ between two types α and β which is defined in Appendix A, section 5, implies that for every interpretation of the language the domain of α is a subset of the domain of β.

[4)] Global rules are used, for instance, in TQA (Petrick, 1982) and EUFID (Burger, 1977). When global rules are a little more complicated than the example above, they raise a completeness problem. It is difficult to know if the rules cover all the cases that can arise. Local rules make completeness issues much more tractable. The rest of this chapter demonstrates that global rules can be avoided more consistently than one might think. In Chapter VI, section 2 I shall take up the issue of local vs. global translation rules again.

To prove that this indeed guarantees the legitimacy of any target language expression resulting from the translation of a legitimate source language expression, the following property which holds for all branching categories of the logical language is needed: if a branching category which constructs an expression of type β out of expressions of type $\alpha, \ldots, \alpha_n$ is applied to expressions of type $\gamma_1, \ldots, \gamma_n$ such that $\gamma_1 \overset{\subseteq}{\overline{T}} \alpha_1, \ldots, \gamma_n \overset{\subseteq}{\overline{T}} \alpha_n$, then the resulting expression has a type δ such that $\delta \overset{\subseteq}{\overline{T}} \beta$.

Because of this property of the branching categories, the property of source language expressions *e* that ET(*e*) is a legitimate target language expression and $\text{TYPE}(\text{ET}(e)) \overset{\subseteq}{\overline{T}} \text{TYPE}(e)$, required for constants and trivially fulfilled for variables, carries over to arbitrary expressions.

To summarize: If a source language SL and a target language TL have identical type systems and have identical branching categories which are "transparent for the type-inclusion relation", a constant-translation from SL to TL is defined as a function CT from the descriptive constants of SL into the expressions of TL such that for every descriptive constant *c* of SL:

$$\text{TYPE}(\text{CT}(c)) \overset{\subseteq}{\overline{T}} \text{TYPE}(c).$$

Such a constant-translation CT induces a function ET on expressions which replaces every constant *c* in the expression by its constant-translation CT(*c*). ET assigns to any SL-expression *e* a TL-expression such that

$$\text{TYPE}(\text{ET}(e)) \overset{\subseteq}{\overline{T}} \text{TYPE}(e).$$

3. **Translation between Languages with different Type Systems.**

In working out the idea of a synonymy-translation just above, it was assumed that the type systems of the source language and the target language were identical. This is a case which actually may occur – for instance, if both languages have only one descriptive type (the type *entity* in the example in the previous section). The case that the type systems of source language and target language are different will now be considered.

If the language used for representing the meanings of questions in a question-answering system has a many-sorted type system, important simplification transformations are possible (see Chapter V, section 5), the efficiency of proof procedures may be improved (Minker, 1978), and it will be possible to test for "semantic anomaly" (Appendix A, section 5).

To take full advantage of the possibilities of many-sorted languages, the type system of the English-oriented formal language and the type system of the data base language must be allowed to be different. At the highest domain-independent level a refined many-sorted type system cannot be used because any assignment of refined types to constants of the language would impose constraints on those constants which are domain-dependent. At the data base level, on the other hand, there are constraints that should be expressed by the type system – e.g. what the domains of applicability of the functions corresponding to the attributes of the data base are.

Because of this discrepancy between the type systems at the highest and the lowest level of the system, it is worthwhile to define synonymy-translations between languages which not only differ in the descriptive constants they contain, but also in the descriptive atomic types they have.

The translation specification as introduced in the previous section can also be used for this case. A list which specifies for every source language constant a synonymous target language expression defines, given a target language interpretation, at most one source language interpretation as being compatible with it: the interpretation which assigns to every descriptive constant of the source language SL the denotation of the corresponding expression of the target language TL, and to every atomic type α of SL the denotation of the TL expression which corresponds to the SL constant GS_α.[5] (To every formal constant or type, the SL interpretation assigns the same entity as the TL interpretation.)

It is not very difficult to define a conversion function H which assigns to any expression *E* of the source language SL an expression H*(E)* of the target language TL, in such a way that both expressions have the same denotation under compatible interpretations of both languages. To take care of the ranges of the variables in the expressions, the definition of H is less trivial than the corresponding definition (of the function ET) in the previous section. To assign types to the variables in the target language expression which "translate" variables of the source language expression, the definition uses a correspondence between the type systems of the two languages which is induced by the constant-translation. Given a constant-translation CT, we define the corresponding *atomic-type-translation* AT as the function which assigns to any atomic type α of SL the type TYPE(CT(GS_α)) of TL.

Given an atomic-type-translation AT, the corresponding *type-translation* TT is defined as the function which assigns to any type τ of SL the type of TL which is obtained by substituting for every atomic type α in τ the type AT(α).

H is recursively defined as follows:

H(e) = $_{\text{def}}$

if e is a constant **then** CT(e)
else if e is a variable **then** VT(e)
else if e has the form (λx: D)
  then (λy: *(if:* $y \in A$, *then:* D')),
    where $y \equiv$ VT(x)
      $A \equiv$ H(GENSET(TYPE(x))
      $D' \equiv$ H(D)
else [e has the form b*(*sel_1*:* e_1, ..., sel_n*:* e_n*)*]
  b*(*sel_1*:* e_1', ..., sel_n*:* e_n'*)*
  where $e_1' \equiv$ H(e_1), ..., $e_n' \equiv$ H(e_n).

[5] We assume that the source-language contains for every atomic type α a constant GS_α denoting the domain of α.

In the definition of H, two functions were used which have not been introduced before: VT and GENSET.

VT may be any function which assigns to any variable v of SL a distinct variable u of TL, such that TYPE(u) = TT(TYPE(v)).

The function GENSET assigns to any type an expression which denotes the domain of that type under any interpretation of the language. It is defined as follows:

GENSET (α) = $_{def}$
if α is an atomic type **then** GS_α
else if α has the form $S(\alpha')$ **then** power (GENSET (α'))
else if α has the form $B(\alpha')$ **then** bags (GENSET (α'))
else if α has the form $F(\alpha')$ **then** files (GENSET (α'))
else if α has the form $L(\alpha')$ **then** lists (GENSET (α'))
else if α has the form $\langle\alpha_1, \ldots, \alpha_n\rangle$
then cartesian-product (GENSET (α_1), ..., GENSET (α_n))
else if α has the form $(\alpha_a \rightarrow \alpha_v)$
then functions$_t$ (*domain:* GENSET (α_a), *range:* GENSET (α_v))
else if α has the form $(\alpha_a \dashrightarrow \alpha_v)$
then functions$_p$ (*domain:* GENSET (α_a),
range: GENSET (α_v))
else if α has the form $(\alpha_1, \ldots, \alpha_n)$
then union (GENSET (α_1), ..., GENSET (α_n))
else if α has the form AMT(β)
then bag-to-set (
(*for:* cartesian-product (union ($GS_{integer}$, GS_{real}), GENSET (β)),
apply: (λx: (*num:* x [1], *unit:* x [2]))))
where x has the type $\langle \cup$ (*integer, real*), $\beta\rangle$
else if α has the form ID_i (β)
then bag-to-set ((*for:* GENSET (β), *apply:* (λy: id_i (y))))
where y has the type β.

(The semantics of the branching categories "power", "bags", "files", "lists", "cartesian-product", "functions$_t$", "functions$_p$", "union" and "bag-to-set" is given in Appendix A.)

In the definition of H, the translation of expressions of the form (λx: *D*) may be refined by giving separate consideration to the case that

H(GENSET (α)) $\equiv$ GENSET (TT(α)), where α is the type of x.

In this case, the simpler expression (λy: *D'*) may be used instead of the translation

(λy: (*if:* $y \in A$, *then:* *D'*)),

because the condition $y \in A$ necessarily has the value TRUE.

To guarantee that, for any expression E, $H(E)$ is a legitimate expression of TL, we require that for any constant c

$$\text{TYPE}(\text{CT}(c)) \subsetneq_{\text{T}} \text{TT}(\text{TYPE}(c)).$$

That this condition has the desired effect follows by the same line of reasoning that was indicated for the corresponding condition in the previous section.[6] It is reasonable to add the following two requirements because they enforce a good "style" of designing translation specifications:

- For any atomic type α of SL holds that all elements of COMPONENTS(AT(α)) are atomic types of TL.[7]
- For any two different atomic types α and β of SL holds:
  $\text{COMPONENTS}(\text{AT}(\alpha)) \cap \text{COMPONENTS}(\text{AT}(\beta)) = \emptyset$.

These requirements guarantee that types which are considered disjoint at the SL level are not mapped onto identical or overlapping types at the TL level. The background of the first requirement is the fact that atomic types are considered to be disjoint with all compound types; the background of the second requirement is that distinct atomic types are mutually disjoint. These properties of the type system are useful because, for instance, they make it possible to simplify expressions containing unwieldy "function-choices" (see Bronnenberg et al. (1980), section 13, for a simple example).

As an example, reconsider the data base in section 1, analysed according to the method discussed in Chapter IV, section 3. The data base is then viewed as specifying the extensions of the following DBL constants:

- GS_{emp} of type S(*emp*) representing the set of all employees
- GS_{dept} of type S(*dept*) representing the set of all departments
- F-EMP-DEPT, of type (*emp* $\longrightarrow$ *dept*) assigning to every employee a department

We can now describe the relation between EFL and DBL by the following rules[8]:

[6] Notice that this requirement on the translation rules can be effectively checked. This makes it possible to do a "syntactic correctness test" on every new set of translation rules that is entered into a system which is under development. Such a test mode makes it possible to detect mistakes in the formulation of translation rules during the development of the system, rather than during test runs of a finished version. This idea, which parallels the idea of compile-time type checks in high-level programming languages, has greatly diminished the amount of debugging effort involved in realizing a faultless implementation of the PHLIQA1 program.

[7] The function COMPONENTS is defined in Appendix A.

[8] Because of the type constraints on translation rules mentioned above, the rule for HAVE as formulated here would not have been allowed in the PHLIQA1 framework. Instead, it would have been formulated as:

HAVE $\Longrightarrow$ ($\lambda u, v$: u = (*fun:* **function-choice** (F-EMP-DEPT, (λp: FALSE)), *arg:* v))

where u and v range over the domain of *emp* $\cup$ *dept*, and p ranges over the domain of *dept*.

In this way it is avoided that a function is translated into a function with a smaller range, which can lead to semantically anomalous expressions. The final result is the same, however, as in the formulation above.

DEPARTMENTS	$\Longrightarrow$	GS_{dept}
EMPLOYEES	$\Longrightarrow$	GS_{emp}
HAVE	$\Longrightarrow$	$(\lambda u,v$: F-EMP-DEPT$(v) = u)$ where u ranges over the domain of *dept* and v ranges over the domain of *emp*
GS_{entity}	$\Longrightarrow$	$\cup\ (GS_{emp}, GS_{dept})$

Consider example question (1), rendered in EFL as (2).

How many departments have more than 100 employees? (1)

$$\textbf{Count}\ (\{x \in \text{DEPARTMENTS} \mid \textbf{Count}\ (\{y \in \text{EMPLOYEES} \mid \text{HAVE}\ (x,y)\}) > 100\}) \quad (2)$$

where x and y range over the domain of *entity*.

Application of the above rules to (2) yields, after λ-reduction:

$$\textbf{Count}\ (\{x' \in GS_{dept} \mid \textbf{Count}\ (\{y' \in GS_{emp} \mid \text{F-EMP-DEPT}(y') = x'\}) > 100\}) \quad (3)$$

where both x' and y' range over the domain of $dept \cup emp$.

The expression (3) may be further simplified into a formula with the same appearance, except that x' is replaced by u', with the type *dept*, and y' is replaced by v', with the type *emp*.

Allowing differences between the type systems of source language and target language was only one step in the development of the full generality of the translation specification method. In the next section the translation between languages which correspond to each other in a still "looser" way will be discussed.

4. **Identification Translations.**

4.1. **The Problem of Compound Attributes.**

If a target language does not have for every constant of the source language an expression considered synonymous to it, the relation between both languages cannot be defined by a synonymy translation. But it is nevertheless possible that in such a case a weaker, but interesting and useful relation exists between the languages. It may be possible to define an *identification translation* from the source language into the target language, in

which a translation assigns to every constant of the source language an expression of the target language which represents a concept that can "do duty" for the concept that the constant represents, without necessarily being the **same** concept. (If it **is** the same concept in every case, the identification translation is in fact a synonymy translation.)

As an example of a situation where this technique is needed, consider a data base which has a file of DEPARTMENTS, and which has NUMBER-OF-EMPLOYEES as an attribute of this file. This data base specifies an interpretation of a logical language which contains the set-constant GS_{dept} and the function #EMP (from departments to integers) as its descriptive constants.

This data base contains sufficient information to answer question (1) rendered in EFL as (2).

"How many departments have more than 100 employees?" (1)

Count ($\{x \in$ DEPARTMENTS $|$
Count ($\{y \in$ EMPLOYEES $|$ HAVE $(x,y)\}) > 100\}$). (2)
(Both x and y range over type *entity*.)

In terms of the new data base just introduced, the query expressed by (1) would be:

Count ($\{x \in GS_{dept} \mid$ #EMP $(x) > 100\}$). (3)

In describing the relation between EFL′ and DBL for this case, a new difficulty arises. The DBL constants do not allow the construction of DBL expressions whose denotations involve employees. So the EFL′ constant EMPLOYEES cannot be translated into an equivalent DBL expression – nor can the relation HAVE, for lack of a suitable domain. This may seem to require giving up local translation for certain cases: instead, one would have to use an algorithm which looks out for sub-expressions of the form (λy: **Count** ($\{x \in$ EMPLOYEES $|$ HAVE $(x,y)\}$)), where y is ranging over DEPARTMENTS, and then translates this whole expression into: #EMP. This would not be attractive – it could only work if EFL′ expressions would be first transformed so as to always contain this expression in exactly this form, or if a large number of corresponding transformations for differently structured expressions would be specified. Happily, the "identification translation" provides another solution.

Although in DBL terms one cannot talk about employees, one can talk about objects which stand in a one-to-one correspondence to the employees:

the pairs consisting of a department d and a positive integer i such that i is not larger than the value of #EMP for d (see *fig. 1.*).

Therefore, employees may be "represented" by such pairs. The most straightforward way to implement this idea would be to translate constants denoting sets of employees into constants denoting sets of pairs ⟨*department, integer*⟩, and functions on employees into functions on such pairs. For the example we are considering, this would lead to the following translation rules:

$$\begin{aligned}
&\text{EMPLOYEES} &&\Longrightarrow \cup\,(\textit{for}\text{: } \mathrm{GS}_{\textit{dept}},\ \textit{apply}\text{: } (\lambda x\text{: } \{x\} \times \textbf{Ints}\,(\#\text{EMP}\,(x)))\\
&\text{DEPARTMENTS} &&\Longrightarrow \mathrm{GS}_{\textit{dept}}\\
&\text{HAVE} &&\Longrightarrow (\lambda u, v\text{: } u = v\,[1])\\
&\mathrm{GS}_{\textit{entity}} &&\Longrightarrow \cup\,(\mathrm{GS}_{\textit{dept}},\ \cup(\textit{for}\text{: } \mathrm{GS}_{\textit{dept}},\\
&&&\qquad\qquad \textit{apply}\text{: } (\lambda x\text{: } \{x\} \times \textbf{Ints}\,(\#\text{EMP}\,(x)))))
\end{aligned}$$

(**Ints** is a function which assigns to any integer i the set of those integers j such that $0 < j \leq i$.)

Application of these rules to (2) yields, after λ-reduction:

$$\begin{aligned}
&\textbf{Count}\,(\{z \in \mathrm{GS}_{\textit{dept}} \mid\\
&\quad \textbf{Count}\,(\{y \in \cup(\textit{for}\text{: } \mathrm{GS}_{\textit{dept}},\\
&\qquad\qquad \textit{apply}\text{: } (\lambda x\text{: } \{x\} \times \textbf{Ints}\,(\#\text{EMP}\,(x))) \mid y\,[1] = z\})\\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad > 100\})
\end{aligned} \tag{4}$$

This expression can in fact be shown to be equivalent to (3) above. In this case, this method yields a correct result. There are problems with the method nevertheless.

Notice that EFL′ individuals are translated into compound DBL entities. This means that in EFL′ it would be necessary to give up the idea that atomic types and compound types have disjoint domains. This may cause technical problems, in that the applicability conditions of some important simplification transformations have to be reconsidered. Another, more serious problem is the fact that the correctness of the method just illustrated depends on global properties of the formula that it is applied to, properties which have not been made explicit. This method therefore can not be guaranteed to work in every case. If the source language formula, for instance, would contain a condition to the effect that a certain employee would equal a certain pair ⟨*department, integer*⟩ (a condition which would be logically equivalent to FALSE), then this condition would be translated into one which for some data bases would come out TRUE.

A more complicated version of the same idea was developed which fares better in dealing with problems of this kind. Instead of representing employees directly by pairs ⟨*department, integer*⟩, they are represented by objects which have a one-to-one correspondence with these pairs – these objects must be disjoint with the domains of all other (atomic or compound) semantic types which are introduced by translation rules not dealing with the source language type *employee*. After introducing this refinement, the translation rules for the example are as follows.

$$
\begin{array}{lll}
\textsc{employees} & \Longrightarrow & (\textit{for}: \cup(\textit{for}: \mathrm{GS}_{\textit{dept}}, \\
 & & \quad \textit{apply}: (\lambda x: \{x\} \times \mathbf{Ints}(\#\textsc{emp}(x))), \\
 & & \textit{apply}: (\lambda y: \mathbf{id}_{\textit{emp}}(y))) \\
\textsc{departments} & \Longrightarrow & \mathrm{GS}_{\textit{dept}} \\
\textsc{have} & \Longrightarrow & (\lambda u, v: u = \mathbf{rid}(v)[1]) \\
\mathrm{GS}_{\textit{entity}} & \Longrightarrow & \cup(\mathrm{GS}_{\textit{dept}}, (\textit{for}: \cup(\textit{for}: \mathrm{GS}_{\textit{dept}}, \\
 & & \quad \textit{apply}: (\lambda x: \{x\} \times \mathbf{Ints}(\#\textsc{emp}(x))), \\
 & & \quad \textit{apply}: (\lambda y: \mathbf{id}_{\textit{emp}}(y))))
\end{array}
$$

$\mathbf{id}_{\textit{emp}}$ is a function which establishes a one-to-one correspondence between its domain and its range (its range is disjoint with all other semantic types); **rid** is the inverse of $\mathbf{id}_{\textit{emp}}$.

Application of these rules to (2) yields (5) which is logically equivalent to (3) above.

$$
\begin{array}{l}
\mathbf{Count}(\{z \in \mathrm{GS}_{\textit{dept}} \mid \\
\quad \mathbf{Count}(\{y \in (\textit{for}: \textsc{depts}, \\
\quad\quad \textit{apply}: (\lambda x: \{x\} \times \mathbf{Ints}(\#\textsc{emp}(x)))), \\
\quad\quad \mid \mathbf{rid}(y)[1] = z\}) \\
\quad > 100\})
\end{array}
\qquad (5)
$$

where x, y and z range over $\cup$ (*dept*, ID_{emp} (⟨*dept, integer*⟩)).
(This is demonstrated in detail in section 5).

4.2. **More Complex Examples.**

The technique just described is not an ad hoc way of treating one specific example but is rather generally applicable. The technique was developed to deal with some quirks of the PHLIQA1 data base, while the example we just went through comes from Moore (1982) who posed the question of how to allow different ways of assessing the cardinalities of sets at the data base level, while treating "how many"-questions in a unified way.

The following example is also from Moore. Consider again the question

How many departments have more than 100 exployees? (1)

but this time with yet another data base which contains not only a file of departments but also a file of offices with attributes indicating the department and the number of employees of each office. Thus, the data base specifies an interpretation of a query-language which contains the set-constants OFFS and DEPTS, and the function-constant F-OFF-DEPT (with type (*off* → *dept*) and #EMP (with type (*off* → *integer*)).[9]

In this case, DBL proxies for employees may be constructed out of pairs consisting of an office and an integer. This leads to these EFL'-to-DBL translation rules:

EMPLOYEES ⟹ ∪(*for:* OFFS
apply: (λ*x*: (*for:* **Ints** (#EMP (*x*)),
apply: (λ*y*: $\mathbf{id}_{emp}$ (⟨*x,y*⟩)))))

DEPARTMENTS ⟹ DEPTS

ENTITIES ⟹ ∪(DEPTS, ...)

HAVE ⟹ (λ *u,v*: F-OFF-DEPT (**rid** (*v*) [1]) = *u*)

When these rules are applied to the EFL representation of (1), i.e. (2), and a simplification process similar to the one described in section 5 is applied next, the final result is (3).

Count ({*x* ∈ DEPARTMENTS | **Count** ({*y* ∈ EMPLOYEES | HAVE (*x,y*)}) > 100}) (2)

Count ({*z* ∈ DEPTS | **Sum** (*for*: {x ∈ OFFS | F-OFF-DEPT (*x*) = *z*},
apply: #EMP) > 100}) (3)

[9] OFFS ≡ GS_{off} and DEPTS ≡ GS_{dept}.

Thus, this example yields a more complicated DBL expression, where the values of the #EMP attribute for the different offices of one department are added up.

It may be noted that the treatment of this example is exactly parallel to the treatment in the previous subsection: in both cases, one specific DBL identification has been constructed for every employee. By the way this identification was constructed, the fact that these sets of employees of departments or offices are disjoint has also been captured.

For instance, in the treatment of the data base of section 4.1, it was implicitly assumed that any employee can only be in one department at once. On the basis of that treatment it is also possible to answer

How many employees do the departments have together? (4)

where the disjointness of the sets of employees of the departments is needed to be able to compute this number. In EFL', (4) might be represented as

Count (∪ (*for*: DEPARTMENTS,
apply: (λ*x*: {*y* ∈ EMPLOYEES | HAVE (*x*,*y*)}))) (5)

Application of the rules of subsection 4.1, followed by appropriate simplifications, yields DBL expression

Sum (*for:* DEPTS, *apply:* #EMP) (6)

One may, however, imagine situations which are partially similar but where the assumption that different departments have different sets of employees does not hold. Then, the treatment just sketched would not apply, though still the answer to question (1) could be obtained from the data base, since the question would still be equivalent to DBL expression (3).

To deal with this situation an essentially wider framework is needed. (See Chapter VI, section 2).

4.3. **The Definition of Identification Translations.**

A more formal description of the notion of an identification translation that was introduced in subsection 4.1 will now be presented. An identification translation from source language SL into target language TL is defined by specifying

- an atomic-type-translation AT, assigning a type of TL to any atomic type of SL, and
- a constant-translation CT, assigning a TL expression to any SL constant.

To capture the relation between the meanings of the constants and types of

SL and TL, AT and CT must fulfill the following intuitive requirements:

1. In every state of affairs of the subject domain, for every atomic type α there is a one-to-one function f_α which maps the extension of the concept represented by α onto a subset of the extension of the concept represented by AT(α). (In a particularly important special case, f_α is simply the identity function). The idea behind this is that at the TL level elements of the domain of AT(α) are used as "proxies" for the elements of the domain of α.
2. Either constant C and expression CT(C) represent the same concept, or C has a descriptive type[10] and C and CT(C) represent "corresponding" concepts: concepts such that replacing every individual X (belonging to the domain of atomic type α) in the extension of C by $f_\alpha(X)$ necessarily yields the extension of CT(C).

To include identification translations, the formal specification of the requirements for a constant-translation CT + atomic-type-translation AT (see section 3) must be modified to read as follows:

1. a. For any atomic type φ of SL:
   $\exists X \in \text{COMPONENTS}(\text{AT}(\varphi))$: ($X$ is an atomic type $\vee$ X has the form $\text{ID}_i(\gamma)$).
   b. For any two distinct atomic types φ and ψ of SL:
   $\text{COMPONENTS}(\text{AT}(\varphi)) \cap \text{COMPONENTS}(\text{AT}(\psi)) = \emptyset \vee$
   $\neg \exists i \exists x \in \text{COMPONENTS}(\text{AT}(\varphi)) \exists y \in \text{COMPONENTS}(\text{AT}(\psi))$
   $\exists \gamma \exists \delta$: $x \equiv \text{ID}_i(\gamma)$ & $y \equiv \text{ID}_i(\delta)$.
   c. For any formal atomic type φ of SL: $\text{AT}(\varphi) \equiv \varphi$.
2. a. For any constant C of SL:
   $\text{TYPE}(\text{CT}(C)) \overset{\subseteq}{\text{T}} \text{TT}(\text{TYPE}(C))$,
   where the type translation function TT is the function which assigns to any type τ of SL a type of TL by replacing in τ every occurrence of any atomic type α by AT(α).
   b. For any formal constant F of SL, $\text{CT}(\text{F}) \equiv \text{F}$.

When the relation between a source language SL and a target language TL is described by an identification translation, the compatible interpretations of SL and TL are defined as follows.

An interpretation I of TL and an interpretation J of SL are compatible iff[11]

- For every atomic type α of SL there is a one-to-one function f_α mapping the elements of $\text{DOM}_J(\alpha)$ onto the elements of $\text{D}_I(\text{CT}(\text{GS}_\alpha))$. ($\text{D}_I(\text{CT}(\text{GS}_\alpha))$ may be equal to $\text{DOM}_I(\text{AT}(\alpha))$, but it may also be a proper subset of it.)
- For every constant C of SL, replacing every individual X (belonging to the domain of atomic type α) in $\text{D}_J(C)$ by $f_\alpha(X)$ yields $\text{D}_I(\text{CT}(C))$.

[10] A descriptive type is a type which contains one or more descriptive atomic types.

[11] We use the notation "$\text{DOM}_I(\alpha)$" for "the domain of type α under interpretation I", and "$\text{D}_I(e)$" for "the denotation of expression e under interpretation I".

The definition of the conversion function H remains the same as in the previous section, except for a precaution about the use of the $\mathbf{id}_i$-branchings:

$H(e) =_{def}$

if e is a constant **then** $CT(e)$
else if e is a variable **then** $VT(e)$
else if e has the form $(\lambda x{:}\ D)$
then $(\lambda y{:}\ (\textit{if:}\ y \in A,\ \textit{then:}\ D'))$,
where $y \equiv VT(x)$,
$A \equiv H(\text{GENSET}\ (\text{TYPE}(x))$,
$D' \equiv H(D)$
else [e has the form $b(sel_1{:}\ e_1, \ldots, sel_n{:}\ e_n)$]
if $\neg\ \exists\ i$: ($b \equiv \mathbf{id}_i$ &
there is a type in the range of AT which has the form $ID_i(\alpha)$)
then $b(sel_1{:}\ e_1', \ldots, sel_n{:}\ e_n')$,
where $e_1' \equiv H(e_1), \ldots, e_n' \equiv H(e_n)$.

H is now a partial function: if an expression E already contains an $\mathbf{id}_i$ branching which is also used in the translation[12], $H(E)$ is not defined. In practice this is not a limitation; it is easy to ensure that the convertors used at the different levels all introduce *different* $\mathbf{id}_i$ branchings (if any). Therefore, H can for all practical purposes be considered as a total function. (The functions AT and CT, which occur in the definition of H, are total functions.)

Perhaps more important is a relaxation in the definition of the *correctness* of a conversion that is needed now. Since expressions with descriptive types may now be translated into non-equivalent expressions, all we can still require is that the conversion function H assigns to any closed SL-expression *E with a formal type* a closed TL expression $H(E)$ with a formal type such that E and $H(E)$ always have identical denotations under compatible interpretations of SL and TL. This requirement is sufficient because the EFL′ representation of a question is always a closed expression with a formal type. The function H as well as the simplification convertor translate closed expressions into closed expressions, and expressions with a formal type into expressions with a formal type. Therefore, the representation of the question at any level is a closed expression with a formal type.

To the presupposition-expressions which may be part of the representation of a question and whose value must be preserved as well, the same reasoning applies.

[12] This is the case if a type of the form $ID_i(\alpha)$, for this particular i, occurs in the range of AT.

5. Simplification Transformations.

A set of local substitution transformations usually turns its argument expression into a considerably bigger expression: it replaces constants by expressions which are often larger.

A translation algorithm which implements an identification conversion function in a direct way generally produces very unwieldy expressions which in fact allow for much simpler paraphrases. To avoid unnecessarily long evaluation times for the DBL query, the EFL-to-DBL translation must be followed by processing step which applies logical equivalence transformations so as to achieve the simplest possible formulation of the DBL query.

Sometimes a DBL expression is not only unnecessarily large, and unnecessarily time consuming when evaluated – an expression may be actually impossible to evaluate as it stands, because it contains constants which have an infinite extension. It is worth trying to design the translation rules in such a way that introducing such constants is avoided, if possible. As an example, consider the rule for translating EMPLOYEES used above in section 4.1:

$$\text{EMPLOYEES} \Longrightarrow (\textit{for:} \cup (\textit{for:} \text{ DEPTS}, \textit{apply:} (\lambda x\textit{:} \{x\} \times \textbf{Ints} (\#\text{EMP } (x))),$$
$$\textit{apply:} (\lambda y\textit{:} \textbf{id}_{emp} (y)))$$

This rule is to be preferred to the alternative formulation

$$\text{EMPLOYEES} \Longrightarrow (\textit{for:} \{x \in \text{DEPTS} \times \text{INTEGERS} \mid 0 < x[2] \leqslant \#\text{EMP } (x[1])\},$$
$$\textit{apply:} (\lambda y\textit{:} \textbf{id}_{emp} (y)))$$

although the latter formulation may be deemed conceptually simpler. The former formulation avoids the introduction of the unevaluable constant INTEGERS in the expression. Instead, the integers which play a role in the denotation of the expression are generated by the function **Ints**, whose value can be effectively computed for any argument.

To give an impression of the kinds of simplification transformations that can be usefully applied, let us now return to example (1)

How many departments have more than 100 employees? (1)

This question may be rendered in EFL′ as

$$\textbf{Count} (\{x \in \text{DEPARTMENTS} \mid \textbf{Count} (\{y \in \text{EMPLOYEES} \mid \text{HAVE } (x,y)\}) > 100\}) \quad (2)$$

where x and y have type *entity*.

Applying the identification translation as specified in section 4.1 yields

$$\begin{aligned}&\textbf{Count}\,(\{z \in \text{DEPTS} \mid \\ &\quad \textbf{Count}\,(\{y \in (\textit{for:} \cup\,(\textit{for:}\ \text{DEPTS},\\ &\qquad\qquad \textit{apply:}\ (\lambda x{:}\ \{x\} \times \textbf{Ints}\,(\#\text{EMP}(x)))),\\ &\qquad\quad \textit{apply:}\ (\lambda w{:}\ \textbf{id}_{emp}\,(w))\\ &\qquad |\textbf{rid}\,(y)\,[1] = z\}) > 100\})\end{aligned}$$

where x, y, z, w have type $\cup(dept, \text{ID}_{emp}\,(\langle\text{dept, integer}\rangle))$ (3)

This expression can be considerably simplified. I shall now show this step by step.
Applying to (3) the rule

$$\{y \in (\textit{for: A, apply: B}) \mid C\} \Longrightarrow (\textit{for:}\ \{z \in A \mid (\textit{fun:}\ (\lambda y{:}C),\ \textit{arg:}\ B(z))\},\ \textit{apply:}\ B)$$

plus constraining the types of the variables in a self-evident way, yields

$$\begin{aligned}&\textbf{Count}\,(\{z \in \text{DEPTS} \mid \\ &\quad \textbf{Count}\,(\{(\textit{for:}\ \{u \in \cup(\textit{for:}\ \text{DEPTS},\\ &\qquad\qquad \textit{apply:}\ (\lambda x{:}\ \{x\} \times \textbf{Ints}\,(\#\text{EMP}(x))),\\ &\qquad\quad \mid \textbf{rid}\,(\textbf{id}_{emp}\,(u))\,[1] = z\},\\ &\qquad \textit{apply:}\ \textbf{id}_{emp})\\ &\quad > 100\})\end{aligned}$$ (4)

where x and z range over *dept*, and u ranges over $\langle dept, integer\rangle$.
$\textbf{id}_{emp}$ is an abbreviation for $(\lambda w{:}\ \textbf{id}_{emp}\,(w))$, where w has the type $\langle dept, integer\rangle$.

Applying to (4) the rules

$$\textbf{rid}\,(\textbf{id}_{\alpha}\,(x)) \Longrightarrow x$$

and

$$\begin{aligned}&\{u \in \cup(\textit{for: A}, \textit{apply: B} \mid C(u)\}\\ \Longrightarrow\ &\cup(\textit{for: A}, \textit{apply:}\ (\lambda x :\{z \in B(x) \mid C(z)\}))\end{aligned}$$

yields

$$\begin{array}{l}\textbf{Count}\,(\{z \in \text{DEPTS} \mid \\ \quad \textbf{Count}\,((\textit{for:}\ \cup(\textit{for:}\ \text{DEPTS}, \\ \qquad\qquad \textit{apply:}\ (\lambda x{:}\ \{q \in \{x\} \times \textbf{Ints}\,(\#\text{EMP}(x)) \mid \\ \qquad\qquad\qquad\qquad q\,[1] = z\})), \\ \qquad\quad \textit{apply:}\ \textbf{id}_{emp})) \\ \quad > 100\})\end{array} \tag{5}$$

Applying to (5) the rule

$$\{q \in A \times B \mid q\,[1] = C\} \Longrightarrow (\{C\} \cap A) \times B$$

yields

$$\begin{array}{l}\textbf{Count}\,(\{z \in \text{DEPTS} \mid \\ \quad \textbf{Count}\,((\textit{for:}\ \cup(\textit{for:}\ \text{DEPTS}, \\ \qquad\qquad \textit{apply:}\ (\lambda x{:}\ (\{z\} \cap \{x\}) \times \textbf{Ints}\,(\#\text{EMP}\,(x)))), \\ \qquad\quad \textit{apply:}\ \textbf{id}_{emp})) \\ \quad > 100\})\end{array} \tag{6}$$

Applying to (6) the rule:

$$\begin{array}{l}(\textit{for:}\ A,\ \textit{apply:}\ (\lambda x{:}\ (\{x\} \cap B) \times C)) \\ \Longrightarrow\quad (\textit{for:}\ A \cap B,\ \textit{apply:}\ (\lambda x{:}\ \{x\} \times C))\end{array}$$

yields:

$$\begin{array}{l}\textbf{Count}\,(\{z \in \text{DEPTS} \mid \\ \quad \textbf{Count}\,((\textit{for:}\ \cup(\textit{for:}\ \text{DEPTS} \cap \{z\}, \\ \qquad\qquad \textit{apply:}\ (\lambda x{:}\ \{x\} \times \textbf{Ints}\,(\#\text{EMP}\,(x)))), \\ \qquad\quad \textit{apply:}\ \textbf{id}_{emp})) \\ \quad > 100\})\end{array} \tag{7}$$

Applying to (7) the rule

$$A \cap \{x\} \Longrightarrow \{x\}$$

if x has type α and A is GS_α,

yields:

$$\begin{aligned}&\textbf{Count}\,(\{z \in \textsc{depts} \mid\\ &\qquad \textbf{Count}\,((\mathit{for}\colon \cup(\mathit{for}\colon \{z\}\\ &\qquad\qquad\qquad \mathit{apply}\colon (\lambda x\colon \{x\} \times \textbf{Ints}\,(\#\textsc{emp}(x)))),\\ &\qquad\qquad \mathit{apply}\colon \mathit{id}_{\mathit{emp}}))\\ &\qquad > 100\})\end{aligned} \tag{8}$$

Applying to (8) the rule

$$(\mathit{for}\colon \{z\}, \mathit{apply}\colon F) \Longrightarrow \{F(z)\}$$

yields:

$$\begin{aligned}&\textbf{Count}\,(\{z \in \textsc{depts} \mid\\ &\qquad \textbf{Count}\,(\mathit{for}\colon \cup(\{\{z\} \times \textbf{Ints}\,(\#\textsc{emp}(z))\}),\\ &\qquad\qquad \mathit{apply}\colon \textbf{id}_{\mathit{emp}})\\ &\qquad > 100\})\end{aligned} \tag{9}$$

Applying to (9) the rule

$$\cup(\{A\}) \Longrightarrow A$$

yields:

$$\begin{aligned}&\textbf{Count}\,(\{z \in \textsc{depts} \mid \textbf{Count}\,(\mathit{for}\colon \{z\} \times \textbf{Ints}\,(\#\textsc{emp}(z)),\\ &\qquad\qquad\qquad\qquad \mathit{apply}\colon \textbf{id}_{\mathit{emp}}\\ &\qquad\qquad\qquad > 100\})\end{aligned} \tag{10}$$

Applying to (10) the rule

$$\textbf{Count}\,(\mathit{for}\colon A, \mathit{apply}\colon \textbf{id}_{\alpha}) \Longrightarrow \textbf{Count}(A)$$

yields:

$$\begin{aligned}&\textbf{Count}\,(\{z \in \textsc{depts} \mid \textbf{Count}\,(\{z\} \times \textbf{Ints}\,(\#\textsc{emp}(z)))\\ &\qquad\qquad\qquad > 100\})\end{aligned} \tag{11}$$

Applying to (11) the rule

$$\textbf{Count}\,(\{z\} \times A) \Longrightarrow \textbf{Count}\,(A)$$

yields

Count ({$z \in$ DEPTS | **Count** (**Ints** (#EMP(z)))
> 100}) (12)

Applying to (12) the rule

Count (**Ints** (A)) $\Longrightarrow A$

yields

Count ({$z \in$ DEPTS | #EMP (z) > 100}) (13)

Expression (13) is identical to expression (3) in section 4.1. above.

The last version of the PHLIQA1 simplification module, designed and implemented by W.J.H.J. Bronnenberg, contained about 100 simplification rules which can be compared in complexity to the rules just demonstrated. The algorithm which attempts to simplify a PHLIQA1 expression by applying these rules is rather complex. Two important problems giving raise to these complexities should be mentioned here.

First of all, some useful "simplification rules" make an expression bigger rather than smaller (for instance, the rule which goes from (3) to (4) above). A cumulative counterproductive effect of such rules should be avoided. Secondly, many rules are not valid if applied to expressions which are possibly denotationless. (For instance, *(fun: (λx: A), arg: B)*, with A not containing x, cannot be reduced to A if B is denotationless under certain interpretations. Establishing that an expression necessarily has a denotation is therefore often a prerequisite to the application of simplification rules.

6. **Extending the Data Base Language.**

Question (1), rendered in EFL as (2), has been used as a standard example in the present chapter.

How many departments have more than 100 employees? (1)
Count ({$x \in$ DEPARTMENTS |
Count ({$y \in$ EMPLOYEES | HAVE (x,y)}) > 100}) (2)

Consider now a slight variation on (1):

How many departments have more than 100 people? (3)

which is represented in EFL as

$$\textbf{Count}\,(\{x \in \text{DEPARTMENTS} \mid \textbf{Count}\,(\{y \in \text{PEOPLE} \mid \text{HAVE}\,(x,y)\}) > 100\}) \qquad (4)$$

In certain contexts, it may be justifiable to treat the notion "person" as coreferential with the notion "employee", and process (3) accordingly, as described in the previous sections. But let us consider the case that the subject-domain which provides the background for the interpretation of the question is somewhat broader than the actual data base, so that "person" and "employee" should be treated as non-synonymous which is needed to be able to answer the questions "Are all employees employed by a department?" with "Yes", but "Are all people employed by a department?" with "I don't know". Also in this case, (3) can be seen to be synonymous with (5), and can thus be given a definite answer on the basis of the data base of section 1.

$$\textbf{Count}\,(\{x \in \text{GS}_{dept} \mid \textbf{Count}\,(\{y \in \text{GS}_{emp} \mid \text{F-EMP-DEPT}(y) = x\}) > 100\}) \qquad (5)$$

In order to account for the translation from (4) into (5) a refinement of our translation method is needed because the method described so far has a problem with this example; although the answer to (3) is determined by the data base, the question as formulated refers to entities which are not represented in the data base, cannot be constructed out of such entities, and do not stand in a one-to-one correspondence with entities which can be so constructed. In order to be able to construct a DBL translation of (4) by means of local substitution rules of the kind previously illustrated, an extended version of DBL is defined called DBL*.

DBL* contains exactly the same constants as DBL, plus a constant NONEMPS, denoting the set of people who are not employees. By means of the semantic type system of the formal language it can be guaranteed that for every interpretation of the language the denotation of NONEMPS is disjoint with the denotation of the DBL-expression EMPLOYEES.

The rules for the EFL-to-DBL* translation are:

DEPARTMENTS $\Longrightarrow$ (*for:* GS_{emp}, *apply:* F-EMP-DEPT)

EMPLOYEES $\Longrightarrow$ GS_{emp}

PEOPLE $\Longrightarrow$ $\cup(\text{GS}_{emp}, \text{GS}_{nonemp})$

HAVE $\Longrightarrow$ ($\lambda y,x$: (*if:* $x \in \text{GS}_{emp}$, *then:* F-EMP-DEPT$(x) = y$, *else:* FALSE))
where y ranges over the domain of *dept*, and x ranges over the domain of $\cup$(*emp, nonemp*)

GS_{entity} $\Longrightarrow$ $\cup(\text{GS}_{dept}, \text{GS}_{emp}, \text{GS}_{nonemp})$

Application of these rules to (4) yields an expression which can be seen to be equivalent to (5).

It should be noted that for a question like (3) the simplification component of the program plays a different role than for a question like (1): the EFL-to-DBL* translation applied to (4) yields an expression containing the unevaluable constant NONEMPS. In order to give a definite answer, the system must eliminate this constant from the expression. (In the case of question (1), the simplification only serves more efficient evaluation.) If the elimination does not succeed, the system may still give a meaningful "conditional answer", however: it translates NONEMPS to $\emptyset$ and prefaces the answer with "if there are no persons other than employees..."

In the next chapter, alternatives to the knowledge representation techniques used in PHLIQA1 are reviewed and an attempt will be made to draw conclusions about the approach presented here.

Chapter VI. Alternative Knowledge Representation Techniques.

1. **Introduction.**

The previous two chapters discussed the knowledge representation methods employed in PHLIQA1: formatted data bases and translation specifications. The present chapter pays some attention to alternative techniques. The discussion in this chapter will be focussed on knowledge representation techniques which can be used in a well-structured system – i.e., which make it possible to asses what knowledge is possessed by a knowledge representation component of the system independently of the algorithmic structure of this component or the components that interface with it. The less well-defined proposals which have come out of the work in Artificial Intelligence will thus not be dealt with here, but some of the more formal kinds of techniques will be focussed on in the differen sections of this chapter.

Most closely connected to the discussions in the previous chapter, is a consideration of some variations on the "translation specifications" method. Section 2 discusses the use of "global rules", which translate compound expressions rather than constants. This method is less reliable than the more restricted "constant-translation method" developed in the previous chapter. It is more powerful however, and there are cases where this power is needed.

Section 3 discusses another generalization of the "translation specification" method: the use of recursive translation rules. This method necessitates a different algorithm for treating "high-level queries": the translation must be interwoven with the evaluation of the translation results.

Section 4 treats an important knowledge representation technique which assumes the employment of a theorem-proving algorithm rather than a translation algorithm: the method of using collections of axioms. Section 5 treats an interesting variant of this method, characterized by the "Closed World Assumption".

Section 6 treats a particularly interesting group of systems, whose designers have attempted to fuse the two approaches that I so far treated as disjoint: systems which use an axiomatic knowledge representation (possibly with the Closed World Assumption) but which make a distinction within their axiom collection which parallels the distinction between conceptual and factual information which is at the heart of the PHLIQA1 approach. Because of this relationship to the PHLIQA1 work, this approach is discussed in somewhat more detail than other ones.

The final section of this chapter summarizes the conclusions which may be drawn from the discussions in the other sections.

2. **Rule Schemes.**

2.1. **Introduction.**

The method of translation specifications, discussed in Chapter V, only used translations of constants and atomic types to specify the translation of the expressions of a source language into expressions of a target language. The present section discusses more general translation methods which allow *global rules* (translating compound expressions) and *rule schemes* (which specify translations of sets of compound expressions which satisfy a certain structural description). Since rule schemes necessarily are "global", and since the use of global rules usually entails the use of rule schemes, these extensions will be discussed together. However, two different cases of using global rules and rule schemes will be distinguished, since they have significantly different properties.

If a particularly restricted kind of logical language is used, such as the first-order predicate calculus, global rule schemes may be used as an alternative "notation" for the "local translation rule method" we expounded in section 2 of the previous chapter. We briefly discuss this variant in the next subsection, and then go on, in section 2.3, to discuss the use of global rules in less constrained situations.

2.2. **Rule Schemes for First-order Languages.**

Consider the situation that a translation is to be specified between a source language and a target language which have the same syntatic constructions and the same type systems, but different constants. If the languages contain neither lambda-abstraction nor a rich repertoire of functional operators (for example, if they are first-order languages), it is usually not possible to specify the translation by means of local rules operating on constants only, in the manner indicated in Chapter V, section 2. It would in general not be possible, for instance, to formulate a target language expression equivalent to a source language function constant, since one can not construct compound expressions denoting functions. Rule schemes operating on compound expressions must therefore be used instead.

In the case of first-order languages it is nevertheless possible to specify translations in a way which corresponds closely to the purely local method proposed in Chapter V, section 2. It is a characteristic property of a first-order language that predicates and other function-constants (if allowed) only occur in one specific position in one specific syntactic construction: they are always applied to arguments. Therefore, definitions of predicates and other functions can be formulated without loss of generality as rule schemes

operating on compound expressions consisting of a predicate or other function and its arguments. Such rule schemes involve an implicit iteration over all the possible arguments of the function.

Where in the local "constant-translation method" one would have had the rule

$$P \Longrightarrow (\lambda x_1, \ldots, x_n: Q),$$

now the rule scheme

$$P\,(x_1, \ldots, x_n) \Longrightarrow Q$$

is used, in which Q stands for an expression which has no other free variables than possibly $x_1, \ldots, x_n$.
An effective translation algorithm applying such rules could have the form [1]:

TRANSLATE $(A) =_{\text{def}}$
if A has the form $P(a_1, \ldots, a_n)$ and there exists a rule scheme RS for $P(x_1, \ldots, x_n)$
then RIGHT-HAND-SIDE (RS) $[x_1 := a_1, \ldots, x_n := a_n]$
else if A is an individual constant or a variable **then** A
    else if A has the form **bc** $(A_1, \ldots, A_n)$
        then MAKE-BC (TRANSLATE (A_1), ..., TRANSLATE (A_n)).

The procedure MAKE-BC, applied to expressions $e_1, \ldots, e_n$, creates an expression with branching category **bc** and sub-expressions $e_1, \ldots, e_n$. This algorithm assumes that at most one rule scheme is applicable to any expression A, and that source language and target language have distinct (not necessarily disjoint) sets of n-place predicates and function symbols, while sharing all individual constants and variables. The variants of the algorithm which are needed under other assumptions can be easily constructed.

Thus, any rule scheme of the kind described above, containing exactly one predicate constant or function constant on the left hand side, can be used by an effective algorithm to eliminate these constants. The rule schemes operate almost locally, involving only the predicate or function constant which is to be eliminated, and its unanalysed sub-expressions.

The step to an essentially global method is made when a rule or rule scheme is used to define more complex source language expressions in terms of target language expressions. The next subsection gives an example of a situation where this step seems to be calles for.

[1] Notation: $A[x_1 := a_1, \ldots, x_n := a_n]$ stands for the expression A in which variable x_1 has been replaced thourghout by expression a_1, ..., variable x_n has been replaced throughout by expression a_n.

2.3. **An Example of the Use of Global Rule Schemes.**

Section 4.1 of Chapter V discussed the problem which is raised for natural language data base interfaces by the occurrence of "compound attributes" in the data base. As an example it used a data base containing a DEPARTMENTS-file with an attribute NUMBER-OF-EMPLOYEES. If one tries to specify the translation between high-level, "English-like" representations of queries and low-level "data base oriented" formulations of queries by means of local rules operating on constants only, a situation like this constitutes an obvious challenge: an attribute like "number of employees" corresponds to a compound expression at the English-oriented level, but not to any constants at that level. In Chapter V, section 4, it was shown how, under certain assumptions, this example and other cases of its kind may be dealt with by local rules, albeit local rules of some complexity.

Now the same example will be discussed under slightly different assumptions. Global rules will now turn out to be difficult to avoid. The example question (1), discussed in Chapter V, section 4, and rendered in EFL′ as (2), will again be used as a point of departure.

How many departments have more than 100 employees? (1)

Count ($\{x \in$ DEPARTMENTS $\mid$ **Count** ($\{y \in$ EMPLOYEES $\mid$ HAVE $(x, y)\}$) $>100\}$) (2)

We try to answer this question using the data base of Chapter V, section 4.1, which has a file of DEPARTMENTS and which has NUMBER-OF-EMPLOYEES as an attribute of that file. This data base thus specifies an interpretation of a logical language which contains a set-constant DEPTS and the function #EMP (from departments to integers) as its descriptive constants.

We now abolish the assumption, made in Chapter V, that all departments have disjoint sets of employees. Without this assumption, certain answers which could be given before cease to be valid. For instance, the question

What is the total number of employees of all departments? (3)

does not have a definite answer any more, though upper and lower bounds are still determined by the data base.

Question (1) above, however, still has a definite answer. Also without the disjointness assumption, (2) is equivalent to (4), which is an evaluable DBL expression.

$$\textbf{Count}\,(\{x \in \text{DEPTS} \mid \#\text{EMP}\,(x) > 100\}) \qquad (4)$$

The problem now is to generate (4), or another evaluable DBL expression, on the basis of EFL expression (2). The translation rules of Chapter V must be replaced by rules which do not incorporate the disjointness assumption.

To this effect, the following strategy may be used. Employees are represented at the DBL level by means of the denotations of expressions of the form $\text{id}_{emp}\,(i)$, where i is an integer. To be able to perform the translation, the intermediate level DBL* is introduced, which has the same constants as DBL, plus an additional function DIDS, which assigns to any department the identification numbers of its employees. Note that the extension of DIDS is unknown. For an expression to be evaluable, DIDS must be eliminated. Two properties of DIDS are important:

- For two distinct departments a and b, DIDS (a) and DIDS (b) are not necessarily disjoint.
- For any department d, **Count** (DIDS(d)) = #EMP (d).

The EFL-to-DBL* translation rules now read as follows:

$$\text{EMPLOYEES} \Longrightarrow \cup(\textit{for}\text{: DEPTS}, \textit{apply}\text{: DIDS})$$

$$\text{DEPARTMENTS} \Longrightarrow \text{DEPTS}$$

$$\text{HAVE} \Longrightarrow (\lambda u, v\text{: } v \in \text{DIDS}\,(u))$$

$$\text{GS}_{entity} \Longrightarrow \cup(\text{DEPTS}, \cup(\text{for: DEPTS, apply: DIDS}))$$

Applying these rules to (2) yields:

$$\textbf{Count}\,(\{x \in \text{DEPTS} \mid \textbf{Count}\,(\{y \in (\textit{for}\text{: DEPTS}, \textit{apply}\text{: DIDS}) \mid y \in \text{DIDS}\,(x)\}) > 100\})$$

which is equivalent to:

$$\textbf{Count}\,(\{x \in \text{DEPTS} \mid \textbf{Count}\,(\{y \in \text{DIDS}\,(x)\}) > 100\})$$

which is equivalent to:

$$\textbf{Count}\,(\{x \in \text{DEPTS} \mid \textbf{Count}\,(\text{DIDS}\,(x)) > 100\})$$

which is equivalent to:

$$\textbf{Count}\,(\{x \in \text{DEPTS} \mid \#\text{EMP}\,(x) > 100\})$$

Note that the last simplification, where **Count** (DIDS (x)) was transformed into #EMP (x), was not a logical equivalence transformation, but involved a descriptive global rule scheme:

$$\textbf{Count}\ (\text{DIDS}\ (x)) \Longrightarrow \#\text{EMP}\ (x)$$

2.4. Theoretical and Practical Aspects of the Use of Global Rule Schemes.

To formulate the meaning expressed by a set of global rule schemes, every rule scheme is reformulated as a single rule. This can always be done by using lambda-abstraction; if lambda-abstraction is not already part of the language under consideration, it can be allowed as part of an extended language for formulating translation rules. For instance, the rule mentioned in the previous subsection,

$$\textbf{Count}\ (\text{DIDS}\ (x)) \Longrightarrow \#\text{EMP}\ (x),$$

is rewritten, by abstracting over the free variables, as

$$(\lambda x\text{: }\textbf{Count}\ (\text{DIDS}\ (x)) \Longrightarrow (\lambda x\text{: }\#\text{EMP}\ (x)).$$

If all rule schemes are reformulated in this fashion, the translation specification is a set of rules, each of which defines a source language expression as synonymous to a target language expression. In model-theoretic terms, the meaning of such a set of rules can be stated as follows: an interpretation I of the source language SL is compatible with an interpretation J of the target language TL, iff for every translation rule $A \Longrightarrow B$ the denotation of SL-expression A under I is the same as the denotation of TL-expression B under J.

While local translation rules always define at most one source language interpretation as compatible with a given target language interpretation, global translation rules do not have this property. They may thus be used to represent "indefinite" information.

One problem which must be faced in devising an algorithm which applies global rule schemes, is that any redundancy in the expressive possibilities of the logical language must be matched with redundancies in the formulation of the rule schemes. For instance, if the rule above is applied to a language which also contains a function-composition operator $\otimes$, **Count** $\otimes$ DIDS can not be translated into #EMPS by an algorithm which simply checks the source language expression for sub-expressions of the form described in the rule scheme. The PHLIQA1 language presents many examples of this phenomenon. For instance, complicated examples can be found in the

previous section. In such cases, the translation rule scheme can only be applied after the expression has undergone simplification transformations.

As we have seen from the example in the previous subsection, for languages like the PHLIQA1 language it would not be feasible to specify for every descriptive rule scheme a wide variety of variants applicable to expressions of different forms. We must be satisfied with specifying rules for the simplest cases, and rely on general simplification algorithms for bringing the source expression is a suitable form. To guarantee effective translation, we then need normalizability theorems about the logical language. For many powerful languages, like the PHLIQA1 language, no normalizability theorems exist.

Since different rule schemes may operate on overlapping parts of an expression, it is difficult to assess which set of cases is covered by a given set of rule schemes. For instance, the rules

$$A(B(x)) \Longrightarrow Q$$
$$B(C(x)) \Longrightarrow R$$

do not eliminate the constants A, B, and C out of the expression

$$A(B(C(a)))$$

though all "patterns" occurring in the expression are covered by the rules.

3. **Definitions within one Language.**

3.1. **Introduction.**

So far, methods have been discussed which define source language expressions in terms of expressions of a distinct target language. This has excluded recursive definitions from consideration.[2] The present section investigates the use of constant-definitions which are allowed to be recursive. In order to make that formally possible, both source language and target language are assumed to be subsets of one language, called the "union-language".

The definitions may all be viewed as axioms of the form $A = E$, where A is the defined constant and E is the defining expression.

Given an interpretation of the target language constants, we define as compatible with this interpretation: all interpretations of the union-language

[2] It must be noted, however, that phenomena which are usually handled by means of recursion can often be handled differently if the logical language is rich enough. For instance, if the language contains a closure-operator, "ancestor" may be translated into "parent$^+$"

which "contain" the target language interpretation and which assign the value TRUE to the axioms. A compatible source language interpretation is a subset of at least one compatible union-language interpretation.

Nothing in this formulation prevents a definition from being recursive. There is no reason why a source language constant could not occur in the right hand side expression of an axiom defining it.

Less crucial advantages follow from the fact that source language and target language may be allowed to overlap so that translating a constant into a synonymous constant may be skipped. In addition, source language and target language need not be required to exhaust the union-language. It is possible therefore to use intermediate steps in the definition of some of the notions, without having to introduce extra "language levels".

The problematic side of abolishing the definite distinction between source language and target language is that it becomes less self-evident whether a given set of rules defines an effective translation. The structure of the formalism no longer guarantees that every set of rules has this property.

What sets of rules can be used also depends on the structure of the algorithm which is to be applied. An algorithm in the PHLIQA-style, which embodies a strict separation between translation and evaluation, imposes more severe constraints than an algorithm in the PLANNER-style, which interweaves these processes. The next subsections discuss these alternatives.

3.2. **Translation and Evaluation.**

Consider a situation where a source language SL and a target language TL both are subsets of a union-language UL. The SL-to-TL translation is specified by means of UL-axioms of the form $C = E$, where C is a UL-constant and E is a UL-expression. A data base specifies values of TL-constants.

To compute values of SL-expressions on the basis of this information, different kinds of algorithms may be used. The present subsection discusses a treatment which makes a clear separation between SL-to-TL translation and TL evaluation.[3] (The next subsection discusses an alternative).

This method would first translate the SL-query into TL, and then evaluate the resulting TL-expression. This can be summarized as follows:

result: = EVAL (TRANSLATE (x))

A possibly more efficient variant would be:

result: = EVAL (SIMPLIFY (TRANSLATE (x))),

[3] This method is akin in style to the PHLIQA1 method, which may be viewed as a refined version of it.

where SIMPLIFY would be a procedure which transforms target language expressions into logically equivalent target language expressions which are easier to evaluate.
EVAL is recursively defined:
EVAL(A) = $_{\text{def}}$
if A has the form 'equal(x_1, x_2)' **then**
if EVAL (x_1) = EVAL (x_2) **then** TRUE **else** FALSE.
else if A has the form 'conj (x_1, x_2)' **then if** EVAL (x_1) = FALSE
then FALSE
else if EVAL (x_2) = FALSE
then FALSE **else** TRUE

else if A has the form 'disj (x_1, x_2)' **then if** EVAL (x_1) = TRUE **then** FALSE
else if EVAL (x_2) = TRUE
then TRUE **else** FALSE

else if A has the form 'application (x_1, x_2)' **then**
if x_1 is a constant **then** the value which the data base function x_1 yields for the argument EVAL (x_2)
else if x_1 has the form $(\lambda y: D)$
then EVAL $(D\,[y := x_2])$

else ...

It is important to notice that when the value of an expression is computed, it is not necessarily the case that the values of all its sub-expressions will be computed in the process. (See the bodies of the procedures for computing the values for conjunctions and disjunctions above.)

TRANSLATE is a "Leibniz-algorithm" which tries to achieve a translation by means of substitution of equivalents.

TRANSLATE (A) = $_{\text{def}}$

if there is an axiom of the form $A = E$ **then** TRANSLATE (E)
else if A has the form 'bc $(A_1, \ldots, A_n)$', where 'bc' is some syntactic construction of the logical language,
then MAKE-BC (TRANSLATE (A_1), ..., TRANSLATE (A_n))
else A.

One important constraint is that no effective translation is possible if some of the rules are recursive, since the translation algorithm would not terminate in

that case. For instance, a constant A is not allowed to occur in expression E if there is an axiom of the form $A = E$ – nor is it allowed to occur in the expression F if there is a constant B in E such that there are axioms $A = E$ and $B = F$.

Recursive definitions can be put to use, however, if the translation from source language into target language is not separated from the evaluation of the target language expressions, as assumed so far. This possibility is explored in the next subsection.

3.3. **Interweaving Translation and Evaluation.**

Translation specifications may be used by a different kind of algorithm than the one indicated in the previous subsection. Translation and evaluation do not have to be separate steps, applied one by one to an expression representing the whole content of the question. Instead, translation and evaluation may be interwoven within one recursive algorithm. In this case, the computation of the value of a source-language expression X is simply described as

EVAL* (X),

where EVAL* is an algorithm which attempts to evaluate a source-language expression directly, only calling the translation module when that is necessary. For instance, an EVAL* algorithm running parallel to the EVAL algorithm of the previous subsection would be defined as follows.

EVAL* $(A) =_{\text{def}}$

if A is a constant and there is an axiom of the form $A = E$ **then** EVAL* (E)
else if A has the form 'equal (x_1, x_2)' **then**
    if EVAL* (x_1) = EVAL* (x_2) **then** TRUE **else** FALSE
else if A has the form 'conj (x_1, x_2)' **then if** EVAL* (x_1) = FALSE
                                then FALSE
                                else if EVAL* (x_2) = FALSE
                                then FALSE **else** TRUE
else if A has the form 'disj (x_1, x_2)' **then if** EVAL* (x_1) = TRUE **then** TRUE
                                else if EVAL* (x_2) = TRUE
                                then TRUE **else** FALSE
else if A has the form 'application (x_1, x_2)' **then**
    if x_1 is a TL-constant **then** the value which the data base function x_1 yields for the argument EVAL* (x_2)
    else if x_1 has the form '$(\lambda y: D)$'
        then EVAL* $(D\,[y := x_2])$
else...

An algorithm of this sort was implemented in the MICROPLANNER system (Hewitt, 1972; Winograd, 1972). An advantage of this algorithm, above the one described in the previous subsection, is that it does not exclude recursive definitions. For instance, a definition like

$$\text{Ancestor} \Longrightarrow (\lambda x,y\text{: Parent } (x,y) \vee (\exists z\text{: Parent } (z,y) \,\&\, \text{Ancestor } (x,z))$$

can be allowed now. The algorithm of the previous subsection would never stop applying this definition: the algorithm sketched above would come to an end, however, for a data base which is finite and which does not contain "loops" in the Ancestor-relation (i.e. occurrences of entities which, under the above definition, are their own ancestors).

Extreme care is necessary in exploiting this possibility, however. The logically equivalent definition.

$$\text{Ancestor} \Longrightarrow (\lambda x,y\text{: } (\exists z\text{: (Ancestor } (x,z) \,\&\, \text{Parent } (z,y)) \vee \text{Parent } (x,y))$$

for instance, would lead to a non-terminating execution of the EVAL* algorithm above.

Another disadvantage of the integration of translation and evaluation is that is is not possible any more to introduce a simplification procedure between the translation and the evaluation step. Complicated translations of the kind described in Chapter V and applied in PHLIQA1 would lead to unnecessarily long computation times because of this. Chapter V also described extensions of the method of translation specifications which hinge essentially on the availability of a powerful simplification procedure: unevaluable constants may be introduced by a translation, and a simplification procedure is then required to eliminate these constants before evaluation takes place. Such techniques are incompatible with the algorithmic structure just described.

4. **Representing Knowledge by Means of Axiom Collections.**

The previous sections of this chapter discussed knowledge representation methods which can be viewed as variations on the PHLIQA1 method that was developed in Chapter V: methods which assume a division between a data base (which specifies the values of certain constants) and a set of definitions of concepts which makes it possible to access this data base through a "higher-level" language. The present section discusses an important knowledge representation method with a rather different character: the representation of knowledge by means of axioms, i.e. by means of formulas of a logical language which are stipulated to be true.

The axioms in the knowledge base of a system correspond to the facts that the system is aware of. They determine what states of the world are

"possible" (i.e. compatible with what the system knows): the states of the world corresponding to the interpretations of the logical language which assign to all the axioms the value TRUE.

The paradigm example of a question-answering system with an axiomatic knowledge base is QA3 (Green, 1969) – a system which is still of more than historical interest. QA3 represents knowledge as well as questions by means of first-order predicate calculus formulas. To find its answers it uses a refutation procedure based on the resolution principle (Robinson, 1965). It answers yes/no questions (represented as closed formulas) and wh-questions in the mention-one reading (represented as open formulas).

For the unrestricted predicate calculus, there are no complete decision procedures, which would be guaranteed to find for any formula either a proof or the decision that no proof is possible. There are only complete "proof procedures": if a formula which the procedure is trying to prove is in fact true in all realizations, the proof will be found in a finite number of steps, but otherwise the procedure may continue forever.

QA3 uses a such complete proof procedure. When answering a yes/no question, it tries to prove the query-formula (and answer "yes") or its negation (to answer "no"). If the proof procedure terminates in both cases without a proof, it answers "unsufficient information" (which means that the system has succesfully established that it does not know the answer). But the proof procedure is not a decision procedure, i.e. it is not guaranteed to terminate. Therefore, the attempt to find an answer may have to be abandoned at some arbitrary point. QA3 says "No proof found" in this case. (Allowing the system to work longer on the query *might* yield an answer, which could be either "yes" or "no" or "unsufficient information".)

Besides yes/no questions, QA3 answers questions of the form: "indicate some entity x such that $P(x)$". Any expression α such that $P(\alpha)$ is TRUE is a possible answer to the latter kind of query. Such expressions may be found by the same proof procedure which is used for answering yes/no questions: the procedure tries to disprove $\neg \exists\, x\, P(x)$ by generating a formula α such that the conjunction of $\neg P(\alpha)$ and the contents of the knowledge base implies a contradiction.

The knowledge representation used by QA3 is completely general in the sense that anything which can be expressed in the (limited) query language can be put in the knowledge base.[4] The use of general purpose predicate calculus proof techniques, which seems to be the natural consequence of this, has some drawbacks however:

[4] All implemented and proposed systems of this kind use the first-order predicate calculus to represent their knowledge. As noted before (Chapter I, section 4), this imposes considerable constraints on the kinds of knowledge that can be represented. Often a very strict version of the first-order predicate calculus is used, which does not allow functions – because of the difficulty which resolution-based theorem-proving algorithms have in dealing with equality.

- Termination of the proof procedure cannot be guaranteed.
- Genuine wh-questions are not handled.
- Equality and set-theoretic notions are handled in an inefficient way.
- The procedure gets increasingly inefficient if the number of axioms increases or the axioms become larger.

5. **The Closed World Assumption.**

Carbonell and Collins (1973) and Collins et al. (1975) first pointed out the importance of distinguishing between situations where the sets referred to in a question are stored completely ("closed worlds") and situations where this is not the case ("open worlds"). In the former case there is much more information implicit in the knowledge base than in the latter. If a property *P* is defined in terms of closed sets which are completely stored, then the question whether an object *A* which does not occur at all in the data base has the property *P* may be answered nevertheless. Questions asking whether *P* holds for all individuals in some set may also be answered, as well as those asking for which number of individuals *P* holds. In the "open world" situation, none of these kinds of questions can be answered.

The knowledge representation method of value specifications, described in Chapter V, is clearly geared towards the "closed world" situation: a value specifies the extension of a query language constant completely. Reinterpreting values as partial specifications would invalidate the claims made earlier about the operation of recursive evaluation algorithms and require rethinking the connection between value specifications and conventional data bases. Nevertheless, the use of the value specification technique is not incompatible with "open world" situations. For that purpose, a particularly simple instance of the translation technique discussed in Chapter V must be used. (See the discussion in Chapter V, section 6.)

The present section will focus on the open vs. closed worlds issue in connection with the axiomatic knowledge representation technique. The situation obtaining here is the opposite of the one obtaining for value specifications: the axiomatic knowledge representation technique automatically implements the open world situation. Sets are represented by means of predicates, and axioms may stipulate about certain individuals that they belong or do not belong to some set. That these various stipulations add up to a complete specification of what the members of the set are, is never a default assumption, though it may of course be explicitly stated by means of another axiom. The axiomatic method, as used for instance in QA3, is therefore completely general.

Often, axiom collections are proposed as being the logical entities implemented by a conventional data base. Here, the simplest interpretation

is again the "open world" interpretation: every record in a file represents an axiom. When a file is known to be complete, it may of course be viewed as implicitly containing negated literals for all the n-tuples not contained in the file.

A related but more complicated technique, involving not just an abbreviation convention but a reinterpretation of the model-theoretic status of the whole data base, is the Closed World Assumption. This technique was described by Reiter (1978a); similar techniques had been used in several systems, such as PLANNER (Hewitt, 1972; Winograd, 1972) and PROLOG (Clark, 1978).

A system which makes the Closed World Assumption answers queries on the basis of an "implicit axiom collection" which is constructed out of its explicit axiom collection in the following way:

1. All the explicit axioms are members of the implicit axiom collection.
2. If a positive literal is not provable on the basis of the explicit axioms, its negation is a member is the implicit axiom collection.

Phrased in semantic terms, the Closed World Assumption amounts to the following way of defining a set of interpretations of the logical language by means of a set of axioms: the interpretations of the logical language which may correspond to the state of the world are those realizations of the axioms which assign the value FALSE to every positive literal which is not a valid formula.

Operating with the Closed World Assumption is difficult, because it introduces a kind of "monism" in the knowledge representation – to understand the meaning of a given axiom the other axioms stored must be taken into account. Some problems with this technique will be illustrated now. These problems follow directly from the fact that positive literals play a special role in the semantics.

If definitions involve negation, and the extensions of the defining predicates are not completely specified in the explicit data base, inconsistency results. As an extremely simple example, imagine a knowledge base consisting of the sole axiom

$$\forall x\colon \mathrm{P}(x) = \neg\, \mathrm{Q}(x) \tag{1}$$

The yes/no question represented by the formula

$$\mathrm{P(a)} \tag{2}$$

would be answered "no" on the basis of this knowledge, since under the Closed World Assumption

$$\neg P(a) \tag{3}$$

is a member of the "implicit axiom collection". The yes/no question represented by the formula

$$Q(a) \tag{4}$$

would be answered "no" as well, since

$$\neg Q(a) \tag{5}$$

is also an "implicit axiom". These answers are inconsistent with the axiom (1) which is in the knowledge base of the system.

The same example illustrates a related phenomenon which is a little disturbing: adding a member of the implicit axiom collection to the explicit axiom collection changes the content of the data base. If, for instance, $\neg P(a)$ is added to the knowledge base, Q(a) will henceforth be answered "yes" rather than "no".

Combined with an arbitrary axiom collection, there is no guarantee that the Closed World Assumption can be made without inconsistency. This was noted also by Reiter (1978a). In order to operate reliably with the Closed World Assumption, one should like to impose syntactic constraints on the axioms which are allowed. Reiter (1978a) gives a suggestion in this direction. He shows that the Closed World Assumption does not affect the consistency of an axiom collection containing Horn clauses only. He therefore proposes to constrain the application of the Closed World Assumption to "Horn data bases".

6. **Theorem Proving and Data Bases.**

6.1. **Introduction.**

A system which uses a set of axioms as its knowledge representation need not necessarily treat this set of axioms as one homogeneous body of knowledge accessed by one uniform proof procedure. The systems to be discussed in the present section combine an axiomatic knowledge representation with a perspective which is in some respects similar to the PHLIQA1 approach. They employ a formatted data base with specific knowledge as well as a separate, more freely structured axiom set containing

general knowledge, definitions of concepts, etc. Because of their orientation towards first-order logic and resolution based proof procedures, there are also marked differences between PHLIQA1 and the systems discussed here.

The systems developed by Reiter (1977, 1978b), Chang (1978) and Minker (1978) divide their axiom collections into two distinct parts, called an "extensional data base" (EDB) and an "intensional data base" (IDB). The purpose of this division parallels the distinction we made in the previous chapter between factual and conceptual information. The extensional data base specifies a specific state of affairs in the world, by means of a more or less homogeneous mass of fully instantiated unit clauses which may possibly be stored in a conventional data base. The axioms which constitute the intensional data base may be more complicated (though their format tends to be restrained, as we shall see). Their purpose is to express the relations which exist between the predicates used in the extensional data base and the predicates which may be used in the higher-level formulations of queries to be answered from that data base.[5)]

This distinction parallels the one made in Chapter V – and in some cases the parallel goes even further. Both Reiter and Chang use their intensional axioms to transform an incoming query into a query (or set of queries) only containing base relations, and which can then be evaluated on an ordinary relational data base.

Some limitations of the systems to be discussed here follow from the fact that they all use the first-order predicate calculus without function symbols as the language for formulating their axioms. For formulating the queries Minker uses predicate calculus while Reiter and Chang use predicate calculus extensions comparable to the LUNAR language. (The limitations following from the use of such languages were discussed in Chapter I, section 4).

Although the view, implicit or explicit in all these proposals, that the ordinary notion of a "data base" should be formalized as a collection of base axioms has been rejected here (see Chapter IV, section 5 for that discussion), the logical and algorithmic properties of the systems which

[5)] The term "intensional data base" is an unfortunate one. It suggests that the knowledge represented by the intensional data base would have a different status than the knowledge represented by the extensional data base – that the IDB axioms would be valid in all possible interpretations of the logical language, whereas the EDB axioms would only be claimed to be valid in the interpretation of the language which corresponds to the actual state of the world. It is not evident that this is what one has in mind here. Reiter's examples include in the IDB definitions of terms ("If teacher *u* teaches course *v* and student *w* is enrolled in *v*, then *u* is a teacher of *w*.") as well as contingent facts ("B teaches all computer science courses."). It may also be noted that, if a strict distinction between the logical status of the IDB and the EDB would be maintained, this could be used for distinguishing between answers which are "necessarily true" and answers which are "contingently true"; none of these proposals uses or mentions this possibility. Therefore it may be concluded that the intensional/extensional distinction parallels the conceptual/factual distinction made in Chapter V.

embody this approach are interesting enough to merit further consideration.

The system proposed by Reiter (1977, 1978b) may be viewed as the paradigm example of this approach. A description of this system will be used as a framework for a discussion of some general theoretical issues raised by the approach. Though importantly different in their details, the systems implemented by Minker (1978) and Chang (1978) embody some of the same basic ideas. Therefore, the theoretical discussion carries over directly to their work.

6.2. **Reiter's Proposal.**

Reiter's system uses the knowledge in its Intensional Data Base to map an incoming query onto a set of queries which are formulated completely in terms of the relations in the Extensional Data Base. The extensional data are accessed by query evaluation algorithms rather than theorem-proving procedures. The overall design thus bears some resemblance to the PHLIQA1 structure.

The system answers yes/no queries and wh-queries expressed by LUNAR-style query-formulas. It employs an Extensional Data Base which is a collection of ground literals[6]. The Intensional Data Base of this system consists of a collection of first-order formulas, each of which must have the form

$$\forall x_1 \in \tau_1 : \ldots \forall x_n \in \tau_n : \; W$$

for $n \geq 0$, where W is any quantifier-free first-order formula containing no function symbols (Reiter, 1978b, p. 151). The set of axioms of the IDB must have a property which guarantees that infinite deduction paths cannot arise. A sufficient condition for this is that there are no recursive axioms.[7]

How the system constructs a mapping from "EFL" queries to "DBL" queries may be illustrated with an example query taken from Reiter (1978b).

Consider a simple fragment of an education domain.

IDB *A teaches all calculus courses.* (1a)

$$\forall z \in \text{Calculus}: \; \text{Teach}(A,z) \tag{1b}$$

[6] See Reiter (1978b, p. 152). Terminology: A literal is an atomic formula or a negated atomic formula. An atomic formula is an application of an n-place predicate to n arguments. A ground formula contains no variables.

[7] I.e., if the IDB is put into clause form, no clauses occur which contain the same predicate in a positive as well as in a negative literal.

B teaches all computer science courses (2a)

$\forall y \in$ CS: Teach (B,y) (2b)

If teacher u teaches course v and student w is enrolled in v, then u is a teacher of w. (3a)

$\forall u \in$ Teacher: $\forall y \in$ Course: $\forall w \in$ Student:
Enrolled *(w,v)* & Teach *(u,v)* $\supset$ Teacher-of *(w,u)* (3b)

EDB Teach (A, P100)
Teach (B, P200)
Teach (C, P300)
Teach (D, H100)
Teach (D, H200)

Enrolled (a, C100)
Enrolled (a, P300)
Enrolled (a, CS100)
Enrolled (b, C200)
Enrolled (b, CS200)
Enrolled (b, CS300)
Enrolled (c, H100)
Enrolled (c, C100)
Enrolled (d, H200)
Enrolled (d, P200)
Enrolled (d, P300)

Teacher = {A, B, C, D}
Student = {a, b, c, d}
Course = {C100, C200, CS100, CS200, CS300, H100, H200, P100, P200, P300}
Calculus = {C100, C200}
CS = {CS100, CS200, CS300}

Consider the query

Who are a's teachers? (4a)

{$x \in$ Teacher | Teacher-of (a,x)} (4b)

To be able to treat a query with a resolution algorithm it must be put in the form:

$$[x, T, P] \tag{5}$$

where x is a variable, T is a type and P is a formula in conjunctive normal form.

(Notation: $[x, T, P]$ denotes the set of elements $i \in T$ such that if x denotes i, P is false under all realizations of the axioms which constitute the EDB and the IDB.)
Thus, (4b) is formulated as

$$[x, \text{Teacher}, \neg \text{Teacher-of}(a,x)] \tag{6}$$

Reiter's algorithm first deals with the intensional data base only, postponing access to the extensional data base to a separate next phase of the process. The query (6) is resolved against intensional data base axiom (3b), which leads to

$$[x, \text{Teacher}, \neg \text{Enrolled}(a,v) \vee \neg \text{Teach}(x,v)] \tag{7}$$

Now the system does not yet try to resolve the literals of (7) against the extensional data base. Instead, all possible resolutions against the intensional data base are done first. To begin with, the rightmost literal is resolved. First against axiom (1b), yielding

$$[x, \{A\}, \neg \text{Enrolled}(a,z_{calc})] \tag{8}$$

Then against axiom (2), yielding

$$[x, \{B\}, \neg \text{Enrolled}(a,y_{calc})] \tag{9}$$

As no more resolutions against the intensional data base are possible, the system enters the next phase of the process. All of the query expressions which have been generated are evaluated – i.e. their value is computed by an algorithm which is not a resolution theorem prover but an "ordinary" recursive evaluation program which views the EDB as an "ordinary" data base. For this purpose, the query expressions are reconverted from their special resolution-oriented format into the more usual format which is more suitable for a recursive evaluation program. Thus, the query-expressions (6), (7), (8), (9) are rephrased as (10)-(13):

$\{x \in \text{Teacher} \mid \text{Teacher-of}(a,x)\}$ (10)
$\{x \in \text{Teacher} \mid \text{Enrolled}(a,v) \& \text{Teach}(x,v)\}$ (11)
$\{x \in \{A\} \mid \exists\ z \in \text{Calculus: Enrolled}(a,z)\}$ (12)
$\{x \in \{B\} \mid \exists\ y \in \text{Calculus: Enrolled}(a,y)\}$ (13)

When evaluated these yield, respectively: $\emptyset$, $\{C\}$, $\{A\}$, and $\{B\}$. The union of these is presented as an answer:

$\{A, B, C\}$ (14)

The full description of this method is given in Reiter (1977).

6.3. **The Exhaustiveness of the Answers in Reiter's System.**

First-order theorem-proving systems applied to question-answering represent a wh-question as a formula with a free variable, and generate answers by finding instantiations of the variable which make the formula true. Reiter's system is no exception: it interprets wh-questions as "mention-one" questions.

Reiter makes a point, however, of returning a representation of the set of **all** answers to such a mention-one question. Investigating the relation between such a "super-answer" to a mention-one reading and the possible answers to the mention-all reading reveals that a mention-one super-answer always corresponds to a correct mention-all answer, but that a maximally informative mention-all answer cannot always be encoded as a mention-one super-answer.

For instance, if the axioms determine a state of the world such that

$\{x \mid P(x)\} = \{b\} \quad \vee \quad \{x \mid P(x)\} = \{d\},$ (15)

the mention-all question with content

$\{x \mid P(x)\}$ (16)

would have the disjunctive answer

$\{b\}$ or $\{d\}$ (17)

The mention-one question

$\langle x \mid P(x) \rangle$ (18)

would have as its only answer:

b or d. (19)

The mention-all answer which can be derived from this mention-one answer is:

{b} or {d} or {b,d} (20)

which is a weaker answer than the one that could be given by a system which would evaluate the mention-all query.

This point is also illustrated by the example in the previous subsection: the question

Who are a's teachers? (4a)

represented as

$\{x \in \text{Teacher} \mid \text{Teacher-of}(a,x)\}$ (4b)

The axiom collection consisting of (1), (2), (3), the type-axioms and the extensional data base does not contain enough information to determine the extension of the set $\{x \in \text{Teacher} \mid \text{Teacher-of}(a,x)\}$.
A, B, and C are apparently in it, but the axiom collection does not determine whether D is or not. So the only correct answer would be a disjunctive one:

{A, B, C} or {A, B, C, D} (21)

The fact that the axioms do not determine whether D is included in the set of a's teachers is not apparent from the set of answers in the mention-one reading of the same question. This set consists of "A", "B", "C", "A or B", "A or C", A or D", "B or C", "B or D", "C or D", "A or B or C", "A or B or D", "A or C or D", "B or C or D", "A or B or C or D". Presenting this whole set is not necessary. It can be properly presented as "A", "B", "C", since all the other answers are derivable from these. Answers which are not implied by other answers are called "minimal answers" by Reiter, and his system in fact presents the set of all minimal answers to a mention-one question.

It may be noted that from the set of answers to the mention-one question, one cannot see whether D *may* be one of a's teachers: if the axiom $\neg$ Teacher-of (a,D) were added, the set of answers to the mention-one question would be the same. This may be remedied by slightly extending

Reiter's system. The extended system would respond to a wh-query of the form $\langle x \mid P(x) \rangle$ by not only giving all minimal mention-one answers, but also indicating the domain of predicate P, and all the minimal answers to the query $\langle x \mid \neg P(x) \rangle$.

Such a system would, on the basis of data base above, give an optimal answer to question (4a). It would specify:

- that A, B and C belong to the type "teacher" and are in the positive extension of the predicate "teacher of a",
- that besides these, D is an object of type "teacher" which is in the domain of the predicate "teacher of a",
- that there are no objects of type "teacher" in the domain of the predicate "teacher of a" which are known to be in the negative extension of this predicate.

This information, taken together, means that

$\{x \subset \text{Teacher} \mid \text{Teacher-of}(a,x)\} = \{A, B, C\}$ or
$\{x \subset \text{Teacher} \mid \text{Teacher-of}(a,x)\} = \{A, B, C, D\}$

It should be noted however, that this extended system is not sufficient to guarantee optimal answers in every situation. The extra information about the domain and the negative extension of the predicate cannot capture the information that certain elements in a disjunctive answer are mutually exclusive, as in the first example given above ((15) - (20)).

6.4. **The Closed World Assumption.**

The previous subsection discussed how "super-answers" to mention-one readings of wh-questions, as construed by Reiter, are related to the answers to mention-all readings of such questions. This discussion assumed the knowledge base of the system to be a set of formulas which are meant as axioms in the ordinary model-theoretical sense: they express that the state of the world corresponds to an interpretation of the logical language which assigns the value TRUE to all axioms.

Elsewhere, Reiter (1978a) describes the possibility of a significantly different use of sets of formulas for knowledge representation: as an axiom set which is meant to function under the Closed World Assumption. If one makes this assumption, the interpretations of the logical language which may correspond to the state of the world must not only assign the value TRUE to all axioms, but must also assign the value FALSE to every positive literal which is not assigned the value TRUE by every realization of the axioms.

The system described in Reiter (1978b) is presented as independent of the Closed World Assumption, and should work for "ordinary" axiom sets just

as well. In view of the serious problems of the Closed World Assumption discussed earlier (section 5), Reiter's proposal was first discussed in combination with ordinary axiom collections, in the previous subsection. Now the combination of Reiter's translation method and the Closed World Assumption will be discussed.

Under the Closed World Assumption, the set of answers to the mention-one reading of a wh-question amounts to the same as the answer to its mention-all reading. All the situations discussed in the previous section which give rise to discrepancies between mention-one super-answers and mention-all answers cannot occur if one makes the Closed World Assumption. In all these situations, the truthvalue of some positive literal was not determined by the data base, which is exactly what the Closed World Assumption excludes. Under this assumption, the data base always functions as an abbreviation for a *complete* data base, which for every positive literal either indicates that it is TRUE or that it is FALSE.

The Closed World Assumption is also needed in order to justify a practice which may be observed in all examples given by Reiter, Chang and Minker in illustrating the use of their systems: they consistently represent definitions as implications. In the example cited earlier ((1a)-(14) above), the following axiom constitutes all the information given about the notion "teacher of":

$$\forall u \in \text{Teacher}: \forall v \in \text{Course}: \forall w \in \text{Student}: \text{Enrolled}(w,v) \;\&\; \text{Teach}(u,v) \supset \text{Teacher-of}(w,u) \qquad (3b)$$

Without the Closed World Assumption, no query of the form Teacher-of (A,B) could ever get a negative answer on the basis of the axioms: there is an axiom which specifies conditions which imply the truth of certain formulas of the form Teacher-of (A,B), but there are no axioms indicating when such a formula would be false.

Minker (1978) and Chang (1978) contain similar examples of implicative axioms, explicitly called "definitions", and clearly meant to establish an identity-relation rather than a definition-relation. In the absence of other information about the predicate in that literal, implication-relations with a positive literal on the right hand side function as identity-relations under the Closed World Assumption.

As pointed out in section 5 of this chapter, operating with the Closed World Assumption means that special measures must be taken to guarantee that the axiom set remains consistent. The only constructive proposal about this problem was formulated by Reiter (1978a), who showed that consistency can be guaranteed by confining the axiom set to Horn clauses. If this restriction is assumed, it turns out that the expressive power of a set of axioms does not exceed the expressive power of a sequence of local translation specifications, as will be shown in the next subsection.

6.5. **A Comparison with the Translation Specification Approach.**

The present subsection shows how the results which are achieved by means of Reiter's theorem-proving techniques may be achieved by means of the translation techniques put forward in Chapter V.

The systems developed by Reiter, Chang and Minker all seem to operate under the Closed World Assumption, and all formulate the connection between a defined relation and the base relations by means of axioms which express an implication between a base formula and a positive literal containing the relation to be defined. For every defined relation R, the data base contains a complete specification of the logically independent formulas that imply $R(x_1, \ldots, x_n)$: a set of axioms of the form

$$\begin{aligned} &B_1 \supset R(x_1, \ldots, x_n) \\ &\ldots\ldots\ldots\ldots\ldots \\ &B_k \supset R((x_1, \ldots, x_n) \end{aligned} \tag{22}$$

where $B_1, \ldots, B_k$ are base formulas, R is a defined relation, $x_1, \ldots, x_n$ are variables. (For the sake of simplicity, "intermediate steps" in the definitions are ignored. Reiter (1978b) has shown that this indeed makes no essential difference: the IDB may always be "compiled" into a set of "direct" definitions.) Operating under the Closed World Assumption, we know that if the list (22) contains all the axioms about R, we know that they actually **define** the extension of R: unless $R(a_1, \ldots, a_n)$ follows from the axioms (given an interpretation of the base relations) it is false. Therefore,

$R(a_1, \ldots, a_n)$ is true iff
$B_1 [x_1 := a_1, \ldots, x_n := a_n]$ or ... or $B_k [x_1 := a_1, \ldots, x_n := a_n]$.

Instead of (22), one could therefore write

$$R(x_1, \ldots, x_n) \equiv B_1 \vee \ldots \vee B_k \tag{23}$$

or

$$R \equiv (\lambda x_1, \ldots, x_n : B_1 \vee \ldots \vee B_k). \tag{24}$$

Thus, the IDB axioms defining R might be formulated as the equivalence (24) rather than the set of implications (22). Such a reformulation actually has advantages: it abolishes the need for the Closed World Assumption, and allows for the use of the substitution of equivalents as an "inference procedure". Such a procedure is simpler than resolution theorem-proving. It

can also be applied to other constants than relations, and does not depend on the restricted first-order character of the logical language.

Let us now see how such a reformulation would work in somewhat more detail. As an example, consider again the data base from Reiter (1978b) that was used in section 6.2.

To construct a set of translation specifications equivalent to Reiter's intensional data base, we distinguish three levels of semantic representation. All levels have the same type system. The descriptive atomic types are: *Teacher, Student, Course.* At the highest level (Level 1) we have at least the constant Teacher-of, with type (⟨*Student, Teacher*⟩ → *truthvalue*).

At the intermediate level (Level 2) we have at least the constants Enrolled', with type (⟨*Teacher, Course*⟩ → *truthvalue*), and Teach', with type (⟨*Teacher, Course*⟩ → *truthvalue*).

At the lowest level (Level 3) we have at least the constants Calculus and CS, both with type S(*Course*), Enrolled with type (⟨*Student, Course*⟩ → *truthvalue*), and Teach with type (⟨*Teacher, Course*⟩ → *truthvalue*).

The generic sets of the atomic types are constants at every level: $GS_{Teacher}$, $GS_{Student}$, GS_{Course}. The constant "a" is a constant of type *Student* at every level.

The extensional data base is a "value specification" (see Chapter IV). It specifies the values of the Level 3 constants.
The translation from Level 1 to Level 2 contains the rule:

$$\text{Teacher-of} \Longrightarrow (\lambda w,u: \exists\ v \in GS_{Course}: \text{Enrolled}'(w,v) \,\&\, \text{Teach}'(u,v))$$

The translation from Level 2 to Level 3 contains the rules:

$$\text{Enrolled}' \Longrightarrow \text{Enrolled}$$
$$\text{Teach}' \Longrightarrow (\lambda p,q: (p = \text{A} \ \&\ q \in \text{Calculus}) \vee (p = \text{B} \ \&\ q \quad \text{CS}) \vee \text{Teach}(p,q))$$

Consider the example query

$$\{x \in GS_{Teacher} \mid \text{Teacher-of}(\text{a},x)\}$$

Application of the Level 1-to-Level 2 translation yields, after λ-reduction:

$$\{x \in GS_{Teacher} \mid \exists\ v \in GS_{Course}: \text{Enrolled}'(\text{a},v) \,\&\, \text{Teach}'(x,v)\}$$

Application of the Level 2-to-Level 3 translation yields after λ-reduction:

$$\{x \in GS_{Teacher} \mid \exists\, v \in GS_{Course}: \text{Enrolled}(a,v)\ \&\ ((x = A\ \&\ v \in \text{Calculus})\ \vee\ (x = B\ \&\ v \in \text{CS}\ \vee\ \text{Teach}(x,v))\}$$

This expression may be evaluated against the extensional data base, yielding

$$\{A, B, C\}$$

This may serve as another illustration of the thesis of Chapter V: if all the possibilities of the translation specification method are exploited, it is sufficient to handle many phenomena for which richer frameworks are usually invoked.

7. **Knowledge Representation for Question Answering: Conclusion.**

In the previous chapters we have shown how formatted data bases may be viewed as "value specifications", and how conceptual knowledge may be stored in the form of "local translation rules". Chapter III has shown how these knowledge representations were employed in the question answering system PHLIQA1, leading to an elegant "multilevel" design with an unusually refined modular structure and precise definitions of the tasks of the different modules.

In the present chapter a number of alternative techniques were reviewed. Now I want to sum up the conclusions from that discussion. In doing so, I shall focus on techniques which use a formatted data base for representing factual knowledge. The practical usefulness of such set-up is firmly established. Many systems ware designed to function as natural language interfaces to a previously given formatted data base; PHLIQA1 is a system of this sort.

In Chapter IV it was already argued extensively that formatted data bases can best viewed as "value specifications". The remaining question then is, how to present the conceptual information of the system, which bridges the gap between the natural language terms and the data base primitives. For this purpose, the method of translation specifications was developed in Chapter V.

An interesting alternative to the method of translation specifications was discussed in section 6 of the present chapter: the use of first-order Horn Clauses to specify a translation from a high-level query into an equivalent set of low-level queries. It was shown, however, that this method has serious drawbacks: either it cannot give adequate answers to mention-all questions, or it involves the problematic Closed World Assumption. It was also shown how translation specifications can be used to capture the information in a

Horn clause axiom set which is used under the Closed World Assumption.

In section 2 of the present chapter an example was given which showed that it may sometimes be useful to extend the translation specification method beyond what was presented in the previous chapter (and implemented in the PHLIQA1 system): global translation rules may sometimes be needed. It was also pointed out, however, that it is worth avoiding global rules when that is possible; local rules have important advantages.

Section 3 considered the use of alternative, PLANNER- like algorithms in conjunction with a variant of the translation specification method. Such algorithms might make it possible to use recursive definitions in certain cases. Algorithms of this sort create a reliability problem, however. It seems worthwhile, therefore, to make the logical language to be used sufficiently powerful, so that there is no need to make definitions recursive.

The general conclusion that may be drawn from the present chapter, therefore, is that the approach developed in the previous chapters is a promising one. It is to be expected, however, that global rules must be introduced when more complicated conceptual information must be represented.

Chapter VII. Conclusion: Design Styles.

One of the myths most tenaciously propagated through the oral culture which constitutes the Artificial Intelligence Community of the world is the idea that the problems tackled in this field have an intrinsic complexity which defies rational design methods. A.I. programs are considered to be so complex that they can not be designed from "behind the desk" but need to grow out of an incremental "trial and error" process which at no stage requires the programmer to have a real grasp of what has been built so far. Problem solving programming languages such as PLANNER and QA4 and knowledge representation languages like KRL and KLONE are designed to support such processes. The result of this sort of interaction between programmer and system is, hopefully, a working program which shares with human intelligence the feature of being mysterious: no one knows how it really works.

This "unstructured programming" method is just as unrealistic and idealistic as any other preconception about the design process. No programs are designed strictly in this way. In fact, the more sucessful programs are constructed according to considerably less fanciful methods. As an ideal, however, this programming myth exerts a real influence – which explains, among other things, the dismissals of formal logic and structured programming one often finds[1] and the emphasis placed in A.I. on very highly interactive and intelligent programming tools.[2]

This book has demonstrated the viability of another approach, at least for the case of a large scale natural language question-answering program. The point of departure for the PHLIQA1 system was a thorough analysis of the problem of computational question-answering followed by a careful design process aimed at separating sub-problems as clearly as possible in order to allocate them to independently operating modules with well defined interfaces. The resulting program displayed possibilities for compile-time checking of program data and separate testing of modules – useful features which are most unusual for an A.I. program. As a consequence, the

[1] See, e.g., Winograd (1972) and Winograd (1974).

[2] Rulifsons's (1971) desiderata for the QA4 programming language constitute a most revealing testimony of this. It is the purpose of QA4 "...to provide a method whereby one can construct programs without having to understand the whole problem or even to have worked out a global structure to the solution process. We expect the programs to grow interactively and to be continually refined and improved. We feel that the programmer has a notion of how the program is to work, but does not understand enough of the notion to write algorithms. If he must express his ideas in standard formal languages the strict formality inhibits his intuition and the ideas are lost. By using QA4, he can express these ideas, ambiguous though they may be." (p. 15)

implementation phase of the final program was unusually unproblematic and fast. The debugging phase was, in fact, almost non-existent.

The approach taken here may thus be seen to have certain affinities with the ideas behind "structured programming"[3]. It is perhaps worth pointing out, therefore, that the actual design process of any large complex program differs considerably from the "structured programming ideal".

Usually, structured programming is discussed in the context of designing algorithms for mathematical or otherwise precisely defined tasks. Developing a structured program is then defined as the decomposition of a "higher-level" program written in terms of the same algorithmic constructions (such as iteration, if-then-else, and the like) as the final "low level" program – but containing unanalysed procedures which are spelled out later in terms of other primitives. For mathematical, arithmetic or business applications, this view of developing a properly structured program may be adequate, because one operates within a well-defined formal framework. So when one talks about "unanalysed primitive procedures" whose task is specified without their algorithm being spelled out, one can tacitly assume that this task definition has a certain kind of preciseness.
We can not make this assumption, however, in designing an A.I. program. The complexity and ill-definedness of the problems attacked by such programs call for extensions and refinements of the function-decomposition technique.

A theoretical investigation of the notions involved in the task of the program has therefore played an important role in the design process. Explicit characterizations of the meanings of questions, answers and data bases were developed, to make it possible to define the tasks of program modules with mathematical precision, independently of the design of the algorithms for carrying out the tasks. Because of this, the work on the design of PHLIQA1 has dovetailed, to some extent, with work on theoretically relevant issues in formal linguistics and data base theory.

During the course of the process of designing a question answering system, the "question-to-answers" function is successively described at three different levels:
- informally
- mathematically, by specifying a formal relation between input and output[4].

[3] Wirth (1971), Dahl et al. (1972), Mills (1972), Bates (1976).

[4] A mathematical definition of a function may be a *non-deterministic program,* for example. (A case in point would be the use of an Augmented Transition Network (Woods, 1970) to define the task of a parser.) But we need not exclude the possibility of explicating the task of a procedure by means of a genuinely non-constructive definition – as when the task of a parser is defined by means of a generative transformational grammar.

- as a program (an effective algorithmic specification of the mapping from inputs to outputs).

The design of a program starts from an informal specification of a function which maps informally defined kinds of input data to informally defined kinds of output data. The design process consists in getting more specific about the data as well as about the function.

Thus, our approach can be seen to combine the idea of function-decomposition as the basic design process (as in Dijkstra, 1972) and the distinction of three different levels of abstraction in the description of the program data (Hoare and Dahl, 1972).

In hindsight, the work reported on here may therefore be seen as a defence and implementation of a particular style of software design for complex problems. An important, though largely implicit, claim of this book, then, is that the "structured design" process which resulted in the PHLIQA1 system could be most fruitfully applied to computational problems beyond the issues of question-answering and natural language modelling treated specifically here.

References.

S. Allén and J.S. Petöfi (eds.): *Aspects of Automatized Text Processing.* Papers in Textlinguistics, **17.** Hamburg: Buske, 1979.

D. Bates (ed.): *Structured Programming.* Infotech State of the Art Report, 1976.

N.D. Belnap and T.B. Steel: *The Logic of Questions and Answers.* New Haven and London: Yale Univ. Press, 1976.

M. Bennett: Demonstratives and Indexicals in Montague Grammar. *Synthese* **39,** 1978, 1-80.

H. Biller and E.J. Neuhold: Semantics of Data Bases: the Semantics of Data Models. *Information Systems,* **3,** 1978, 11-30.

D.G. Bobrow and A. Collins (eds.): *Representation and Understanding.* New York: Academic Press, 1975.

D.G. Bobrow and T. Winograd: *An Overview of KRL, a Knowledge Representation Language.* A.I. Memo/Computer Science Dept. Report. Stanford University. Palo Alto, CA: 1976.

E.J. Borowski: English and Truth Functions. *Analysis* **36,** 1976.

W.J.H.J. Bronnenberg, H.C. Bunt, S.P.J. Landsbergen, R.J.H. Scha, W.J. Schoenmakers and E.P.C. van Utteren: The Question Answering System PHLIQA1. In: L. Bolc (ed.): *Natural Language Question Answering Systems.* London: MacMillan, 1980, 217-305.

J.F. Burger: Data Base Semantics in the EUFID System. Paper presented to the *2nd Berkeley Workshop on Distributed Data Management and Computer Networks, May 25-27.* Berkeley, CA: 1977.

J.R. Carbonell and A.M. Collins: Natural Semantics in Artificial Intelligence. *Proceedings of the Third International Joint Conference on Artificial Intelligence,* 1973, 344-351. Reprinted in *AJCL* **1,** Microfiche 3, 1974.

R. Carnap: *Meaning and Necessity.* Chicago: University of Chicago Press, 1947.

C.L. Chang: DEDUCE 2: Further Investigations of Deduction in Relational Data Bases. In: Gallaire and Minker (1978), 201-236.

E. Charniak and Y. Wilks: *Computational Semantics.* Amsterdam: North Holland, 1976.

L. Churchill: *Questioning Strategies in Sociolinguistics.* Rowley, Mass.: Newbury House, 1978.

K.L. Clark: Negation as Failure. In: Gallaire and Minker (1978), 293-324.

CODASYL Data Base Task Group: *April '71 Report.* New York: ACM, 1971.

E.F. Codd: A Relational Model of Data for Large Shared Data Banks. *CACM* **13**/6, 1970, 377-387.

P.R. Cohen and C.R. Perrault: A Plan-Based Theory of Speech Acts. *Cognitive Science,* **3**/3, 1979, 177-212.

K.M. Colby, F.D. Hilf, S. Weber and H.C. Kraemer: Turing-Like Indistinguishability Tests for the Validation of a Computer Simulation of Paranoid Processes. *Artificial Intelligence* **3,** 1972, 199-222.

K.M. Colby: *Artificial Paranoia.* New York: Pergamon, 1975.

A. Collins, E.H. Warnock, N. Aiello, and M.L. Miller: Reasoning from Incomplete Knowledge. In: Bobrow and Collins (1975), 383-416.

O.-J. Dahl, E.W. Dijkstra and C.A.R. Hoare: *Structured Programming.* London and New York: Academic Press, 1972.

F.J. Damerau: Automated Language Processing. In: Williams (1976).

C.J. Date: *An Introduction to Database Systems:* Redding, MA: Addison-Wesley, 1975.

E.W. Dijkstra: Notes on Structured Programming. In: Dahl et al. (1972), 1-82.

L. Dutens (ed.): *G.G. Leibnitii Opera Omnia,* I-VI. Berlin, 1840.

U. Egli and H. Schleichert: Bibliography of the Theory of Questions and Answers. In: Belnap and Steel (1976), 155-198.

E.A. Feigenbaum and J. Feldman (eds.): *Computers and Thought.* New York: McGraw-Hill, 1963.

H. Gallaire and J. Minker (eds.): *Logic and Data Bases.* New York and London: Plenum Press, 1978.

D. Gallin: *Intensional and Higher-order Modal Logic.* Amsterdam: North Holland, 1975.

E.N. Goody: Towards a Theory of Questions. In: E.N. Goody (ed.): *Questions and Politeness. Strategies in Social Interaction.* Cambridge, UK: Cambridge U. Press, 1978, 17-43.

C.C. Green: Theorem-proving by Resolution as a Basis for Question-Answering Systems. In: B. Meltzer and D. Michie (eds.): *Machine Intelligence* **4**. New York: American Elsevier, 1969, 183-205.

J.J. van Griethuysen (ed.): *Concepts and Terminology for the Conceptual Schema and the Information Base.* ISO/TC97/SC5 – N695. New York: American National Standards Institute, 1982.

J. Groenendijk and M. Stokhof (eds.): *Proceedings of the Second Amsterdam Colloquium on Montague Grammar and Related Topics.* January 1978. Amsterdam: Universiteit van Amsterdam.

J. Groenendijk and M. Stokhof: Semantics of *wh*-complements. In: J.A.G. Groenendijk, T.M.V. Janssen and M.B.J. Stokhof (eds.): *Formal Methods in the Study of Language.* Part 2. Amsterdam: Mathematisch Centrum, 1981, 153-182.

J. Groenendijk and M. Stokhof: Semantic analysis of *Wh*-complements. *Linguistics and Philosophy,* **5**/2, 1982, 175-234.

B.J. Grosz: Transportable Natural-Language Interfaces: Problems and Techniques. *Proceedings of the 20th Annual Meeting of the Association for Computational Linguistics.* Toronto, June 1982, 46-50.

C.L. Hamblin: Questions *Australasian J. Phil.* **36,** 1958, 159-168.

C.L. Hamblin: Questions in Montague English. *Foundations of Language,* **10**, 1973, 41-53.

D. Harel: Review of H. Gallaire and J. Minker: Logic and Databases. *Computing Reviews,* **21**/8, 367-369.

R.R. Hauser: Surface Compositionality and the Semantics of Mood. In: Searle et al. (1980), 11-96.

G.G. Hendrix, E.D. Sacerdoti, D. Sagalowicz and J. Slocum: Developing a Natural Language Interface to Complex Data. *ACM TODS* **3**/2, 1978, 105-147.

G.G. Hendrix and W.H. Lewis: Transportable Natural-Language Interface to Databases. *Proc. 19th Annual Meeting of the ACL.* June 29 – July 1, 1981. Stanford, CA: Stanford U., 159-165.

B. Henisz Thompson: *REL English for the User.* Pasadena: California Institute of Technology, 1978.

B. Henisz Thompson and F.B. Thompson: Rapidly Extendable Natural Language. *Proceedings of the 1978 Annual Conference of the ACM,* Vol. I, Washington, Dec. 4-6, 1978, 173-182.

C. Hewitt: Procedural Embedding of Knowledge in PLANNER. *Proceedings of the 2nd International Joint Conference on Artificial Intelligence,* London, 1971, 167-184.

C. Hewitt: *Description and Theoretical Analysis (Using Schemata) of PLANNER.* A.I. Memo No. 251. MIT Project MAC. April 1972.

C.A.R. Hoare and O.-J. Dahl: Notes on Data Structuring. In: Dahl et al. (1972), 83-174.

B. Housel, V. Waddle and S.B. Yao: Functional Dependency Model for Logical Data-Base Design. *Preprints of the Fifth International Conference on Very Large Data Bases.* Rio de Janeiro. October 3-5, 1979, 194-208.

H. Ishiguro: *Leibniz's Philosophy of Logic and Language.* Ithaca, NY: Cornell U. Press, 1972.

A.K. Joshi, B.L. Webber and I. Sag (eds.): *Elements of Discourse Understanding.* Cambridge, UK: Cambridge University Press, 1981.

L. Karttunen: Syntax and Semantics of Questions. *Linguistics and Philosophy,* **1,** 3-44, 1977.

W. Kneale and M. Kneale. *The Development of Logic.* London: Oxford U. Press. 1962.

K. Konolige: *A Framework for a Portable Natural-Language Interface to Large Data Bases.* Menlo Park, CA: SRI AI Center, Technical Note 197, 1979.

K. Konolige: *The Database as Model – a Model Theoretic Approach.* Menlo Park, California: SRI International, Technical Note 255. September, 1981.

R. Kowalski: Logic for Data Description. In: Gallaire and Minker (1978). 77-106.

R. Kowalski and D. Kuehner: Linear Resolution with Selection Function. *Artificial Intelligence* **2**,3/4, 1971, 227-260.

M. Lacroix: *Networks as Internal Structures for Relations.* Brussels: MBLE Technical Note N 121. October, 1977.

S.P.J. Landsbergen: Establishing the "Soundness" of a Question-Answering System. (Abstract). *Proceedings of the 7th International Conference on Computational Linguistics.* Bergen, Norway: 1978.

S.P.J. Landsbergen and R.J.H. Scha: *Formal Languages for Semantic Representation.* Philips Research Labs, MS 10.259. Eindhoven: 1977. Reprinted in: Allén & Petöfi (1979), 59-111.

G.W. Leibniz: Letter to Placcius, 16 November 1686. In: Dutens (1840), Vol. VI, Part I, 32.

P. Medema: A Control Structure for a Question Answering System. *Proceedings of the 4th International Joint Conference on Artificial Intelligence.* Vol. 2. Tbilisi, USSR, 1975.

P. Medema, W.J. Bronnenberg, H.C. Bunt, S.P.J. Landsbergen, R.J.H. Scha, W.J. Schoenmakers, E.P.C. van Utteren: PHLIQA1: Multilevel Semantics in Question-Answering. *AJCL Microfiche* 32, 1975.

H.D. Mills: Mathematical Foundations for Structured Programming. Yorktown Hts, NY: IBM, 1972.

J. Minker: An experimental relational data base system based on logic. In: Gallaire and Minker (1978), 107-49.

R. Montague: English as a Formal Language. In: Visentini et al. (1970), 189-224.

R. Montague: The Proper Treatment of Quantification in Ordinary English. In: J. Hintikka, J. Moravcsik and P. Suppes (eds.): *Approaches to Natural Language. Proceedings of the 1970 Stanford Workshop on Grammar and Semantics.* Dordrecht: D. Reidel, 1973, 221-242.

R. Montague: *Formal Philosophy.* New Haven and London: Yale University Press, 1974.

R.C. Moore: Natural-language access to databases – theoretical/technical issues. *Proceedings of the 20th Annual Meeting of the ACL*, Toronto, 1982, 44-45.

M. Nicolas and H. Gallaire: Data Base: Theory vs. Interpretation. In: Gallaire and Minker (1978), 33-54.

W.H. Paxton: *A framework for speech understanding.* SRI Artificial Intelligence Center, Technical Note 142. Menlo Park, CA: 1977.

S.R. Petrick: Semantic Interpretation in the REQUEST System. In: A. Zampolli (ed.): *Computational and Mathematical Linguistics: Volume II, Proceedings of the 5th International Conference on Computational Linguistics, Pisa, 1973.* Firenze: Casa Editrice Olschki.

S.R. Petrick: On Natural Language Based Computer Systems. *IBM Journal of Research and Development,* **20**/4, 1976, 314-325.

W. Plath: REQUEST: A Natural Language Question-Answering System. *IBM Journal of Research and Development,* **20**/4, 1976, 326-335.

T.C. Potts: Montague's Semiotic: A Syllabus of Errors. *Theoretical Linguistics,* **3**,1/2, 1976.

A. Prior and M. Prior: Erotetic Logic. *Philosophical Review* **64,** 1955, 43-59.

R. Reiter: *An Approach to Deductive Question-Answering.* Cambridge, MA: BBN Report No. 3649. September 1977.

R. Reiter (1978a): On Closed World Data Bases. In: Gallaire and Minker (1978), 55-76.

R. Reiter (1978b): Deductive Question-Answering on Relational Data Bases. In: Gallaire and Minker (1978), 149-178.

J.A. Robinson: A Machine Oriented Logic Based on the Resolution Principle. *JACM* **12,** 1965, 25-41.

J.F. Rulifson: *QA4 Programming Concepts.* Menlo Park, CA: SRI Artificial Intelligence Group, Technical Note 60, August 1971.

R.J.H. Scha: Semantic Types in PHLIQA1. *Proceedings of the 6th International Conference on Computational Linguistics,* Ottawa, 1976.

R.J.H. Scha: Two Logical Views on Data Bases (Abstract). *Workshop "Logic and Data Bases".* Toulouse: ONERA-CERT. November 1977.

R.J.H. Scha: Distributive, Collective and Cumulative Quantification. In: J.A.G. Groenendijk, T.M.V. Janssen and M.B.J. Stokhof (eds.): *Formal Methods in the Study of Language.* Part 2. Amsterdam: Mathematisch Centrum. 1981, 483-512.

R.J.H. Scha: English Words and Data Bases: How to Bridge the Gap. *Proc. of the 20th Annual Meeting of the Association for Computational Linguistics,* 16-18 June 1982, University of Toronto, Toronto, Canada.

L.K. Schubert: Extending the expressive power of semantic networks. *Artificial Intelligence* **7,** 1976, 163-198.

J. Searle: *Speech Acts.* London: Cambridge U. Press. 1969.

J.R. Searle, F. Kiefer and M. Bierwisch (eds.): *Speech Act Theory and Pragmatics.* Dordrecht: Reidel, 1980.

E. Stenius: *Wittgenstein's Tractatus.* Oxford: Basil Blackwell and Mott Ltd, 1960.

A. Tarski: Der Wahrheitsbegriff in den Formalisierten Sprachen. *Studia Philosophica* **1,** 1936, 261-405.

F.B. Thompson and B. Henisz-Thompson: Practical Natural Language Processing. In: M. Rubinoff and M.C. Yovits (eds.): *Advances in Computers,* Vol. 13. New York: Academic Press, 1975, 109-167.

P. Tichý: Questions, Answers and Logic. *Am. Phil. Quart.* **15,** 1978, 275-284. [1978a].

P. Tichý: Two Kinds of Intensional Logic. *Epistemologia* **1,** 1978, 143-146. [1978b].

A.M. Turing: Computing Machinery and Intelligence. *Mind* **59,** 1950, 433-460. Reprinted in: Feigenbaum and Feldman (1963), 11-35.

B. Visentini et al.: *Linguaggi nella Società e nella Tecnica.* Milan: Edizioni di Comunità, 1970.

W. Wahlster, A. Jameson and W. Hoeppner: Glancing, Referring and Explaining in the Dialogue System HAM-RPM, *AJCL,* Microfiche 77, 1978, 53-67.

D.E. Walker (ed.): *Speech Understanding Research.* Final Technical Report. Menlo Park, CA: SRI, 1976.

D. Waltz: Natural language access to a large data base: an engineering approach. *Advance Papers of the Fourth International Joint Conference on Artificial Intelligence.* Tbilisi, Georgia, USSR. September 1975. 868-872.

R. Whately: *Elements of Logic.* 1826. 9th ed. London: John W. Parker, 1848.

M.E. Williams (ed.): Annual review of information science and technology. *Am. Soc. of Inf. Sci.,* **11,** 1976.

T. Winograd: *Understanding Natural Language.* New York: Academic Press, 1972.

T. Winograd: *Five Lectures on Artificial Intelligence.* A.I. Memo no. 246. Computer Science Department Report STAN 74-459. Stanford University, 1974.

N. Wirth: Program Development by Stepwise Refinement. *Comm. ACM* **14/** 4, 1971.

L. Wittgenstein: *Tractatus Logico-Philosophicus.* London: Routledge, 1922.

W.A. Woods: Procedural Semantics for a Question-Answering Machine. *AFIPS Conference Proceedings,* **33,** 1968, 457-471.

W.A. Woods: Transition Network Grammars for Natural Language Analysis. *Comm. ACM,* **13,** 1970.

W.A. Woods, R.M. Kaplan and B. Nash-Webber: *The Lunar Sciences Natural Language Information System: Final Report.* Cambridge, MA: BBN, 1972.

W.A. Woods: *Meaning and Machines.* BBN Report no. 2677. Cambridge, MA: BBN, 1973.

W.A. Woods: Procedural Semantics as a Theory of Meaning. In: Joshi et al. (1981), 300-334.

Appendix A. Syntax and Semantics of the PHLIQA1 Languages.

1. Introduction.

The PHLIQA1 languages are *typed languages*. A system of *semantic types* plays an important role in their syntax and in their semantics. A semantic type may be an *atomic type*, or a *compound type* constructed from atomic types.

The *syntax* of a language defines which expressions belong to it. The definition of the expressions of a PHLIQA1 language consists of two parts:
- a specification of the primitive expressions (the *terms*) of the language.
- a recursive definition of *complex expressions* in terms of simpler component expressions.

The syntax also assigns a semantic type to every expression of the language. The rules which construct compound expressions out of simpler component expressions impose conditions on the semantic types of the component expressions.

The *semantics* of a language specifies how *interpretations* of the language are defined. An interpretation of a PHLIQA1 language is defined in two steps:
1. To every atomic type, a set of entities (called a *domain*) is assigned. (The type system has semantic rules which define the domain of *any* type in terms of the domains of the atomic types).
2. To every term, a *denotation* is assigned; this denotation must be an element of the domain of the type of the term. The semantic rules of the language define the denotation of any expression in terms of the denotations of the terms occurring in it.

2. The Type System of the PHLIQA1 Languages.

A PHLIQA1 language is defined in two steps. First we define a *type system:* a class of *semantic types* and a function (called COMPONENTS) which operates on types. The type system is then used in the definition of the class of expressions of the language.

In the present subsection we give a syntactic definition of the types of the PHLIQA1 languages, and an informal discussion of their semantic aspects. The types are defined by the following rules:
1. Every *atomic type* is a type. There are two kinds of atomic types: *formal* ones and *descriptive* ones. The formal atomic types are *truthvalue*, *integer*, *real* and *string*. They occur in the type system of every PHLIQA1

language. The descriptive atomic types are specified separately for every particular PHLIQA1 language.

2. If α is a type, then $S(\alpha)$, $B(\alpha)$, $L(\alpha)$, $F(\alpha)$ are types.
3. If $\alpha_1, \ldots, \alpha_n$ are types then $\langle \alpha_1, \ldots, \alpha_n \rangle$ is a type.
4. If α and β are types, then $(\alpha \rightarrow \beta)$ and $(\alpha \dashrightarrow \beta)$ are types.
5. If α is a type, AMT (α) is a type.
6. If i is an integer or a string and α is a type, then $ID_i(\alpha)$ is a type.
7. If $\alpha_1, \ldots, \alpha_n$ are types, then $\cup(\alpha_1, \ldots, \alpha_n)$ is a type.

In the definition of the expressions of the language we use a function, called COMPONENTS, which assigns to any type a set of types (its "component types") as follows:

COMPONENTS $(\alpha) =_{def}$
if α has the form $\cup(\alpha_1, \ldots, \alpha_n)$
then COMPONENTS $(\alpha_1) \cup \ldots \cup$ COMPONENTS (α_n)
else if α has the form $\langle \alpha_1, \ldots, \alpha_n \rangle$
then $\{\langle \varphi_1, \ldots, \varphi_n \rangle \mid \forall i : \varphi_i \in$ COMPONENTS $(\alpha_i)\}$
else $\{ \alpha \}$.

The role of the semantic types in the semantics of the language will be precisely described in section 4. But in order to give an intuitive idea about the use of the types, we already give an informal description at this point.

An interpretation of the language assigns to every type a set of entities as its *domain*. The domains of the atomic types are disjoint sets of individual entities. Formal atomic types have the same domain in every interpretation of the language, whereas descriptive atomic types may have different domains assigned to them in different interpretations.

Types of the form $S(\alpha)$, $B(\alpha)$, $F(\alpha)$ or $L(\alpha)$ have domains consisting of compound entities (sets, bags, files and lists, respectively) which consist of elements of the domain of α. (Bags are unordered collections where elements can have multiple occurrences. Files are ordered sets. Lists are ordered and allow multiple occurrences). Types of the form $\langle \alpha_1, \ldots, \alpha_n \rangle$ have domains consisting of n-tuples whose first element is from the domain of $\alpha_1, \ldots$, and whose n^{th} element is from the domain of α_n. Types of the form $(\alpha \rightarrow \beta)$ have domains consisting of total functions from the domain of α into the domain of β; types of the form $(\alpha \dashrightarrow \beta)$ have domains consisting of partial functions from the domain of α into the domain of β.

Types of the form AMT (α) have domains consisting of "amounts", i.e. pairs consisting of a number and an element of the domain of α which is used as a "unit". A type of the form $ID_i(\alpha)$ has a domain consisting of specially constructed objects, which have a one-to-one correspondence to the objects in the domain of α. The domain of $\cup(\alpha_1, \ldots, \alpha_n)$ is the union of the domains of $\alpha_1, \ldots, \alpha_n$.

The type system is used in the syntax which defines the expressions of a PHLIQA1 language (see section 3). Such a syntax also assigns a semantic type to each of the expressions of the language. Syntax and semantics are defined in such a way that for every interpretation of the language the denotation of any expression is an element of the domain of the type of expression.

3. The Definition of the Expressions of a PHLIQA1 Language.

The definition of the expressions of a PHLIQA1 language consists of two parts:
- a specification of the primitive expressions (the *terms*) of the language.
- a recursive definition of complex expressions in terms of simpler component expressions.

There are two kinds of terms: *constants* and *variables*.

There are two kinds of constants: *formal* constants and *descriptive* constants. The formal constants are the same in all the PHLIQA1 languages. They stand for logical or mathematical notions, and receive the same standard denotation for every interpretation of the language. The formal constants are:
- TRUE and FALSE, both with type *truthvalue*.
- the decimal representations of the integers, with type *integer*,
- all alphanumeric strings between quotes, with type *string*.

The descriptive constants are different for every PHLIQA1 language. For every atomic type α, there is a constant GS_α.

For every type, there are countably many variables with this type. Every expression of a PHLIQA1 language is either a *term* (a constant or a variable) or a *complex expression*. Every complex expression has the form **b**$(sel_1: e_1, \ldots, sel_n: e_n)$, where $e_1, \ldots, e_n$ are expressions of the language. **b** is called the *branching category* of the expression; $sel_1, \ldots, sel_n$ are the *selectors* belonging to this branching category.

The recursive definition of the complex expressions of a PHLIQA1 language consists of a number of rules. Each of these rules has the following form: If $e_1, \ldots, e_n$ are expressions of the language, and their types fulfill certain conditions, then **b**$(sel_1: e_1, \ldots, sel_n: e_n)$ is also an expression of the language, and its type can be derived from the types of $e_1, \ldots, e_n$ in a specific way.

PHLIQA1 expressions can also be represented as trees, with constants and variables as terminals, branching categories on the non-terminal nodes, and selectors as labels on the arcs. Such a tree representation corresponds closely to the internal computer representation of the expression. Describing the

expressions as trees of this kind [1] also makes it easier to describe the syntax and the semantics of the languages in a completely formal way – because the trees explicitly display their syntactic structure, and their semantic structure coincides with that. (In section 4 we shall see that the recursive definition of the denotation of an expression parallels the syntactic definition of the expressions given below).

We shall now define a fragment of the PHLIQA1 languages, by giving some (but not all) of the syntactic rules.[2] We omit the elements of the definition which are different for the different languages: the specification of the descriptive atomic types and the descriptive terms. To make the presentation not unnecessarily cumbersome, the type requirements for the sub-expressions of a branching are in many cases chosen to be more simple than in the actual PHLIQA1 system. Especially, the possibility of subexpressions denoting "collections" other than sets and *n*-tuples is largely ignored. To shorten the formulations, we use, for instance, "A has type α" for "A is an expression of the language with type α".

First of all, we have the core operations of the λ-calculus, λ-abstraction and function-application:

1. If *x* is a variable of type α and *B* has type β, **abstraction** *(var: x, descr: B)* has type $(\alpha \dashrightarrow \beta)$. The expression may be abbreviated as $(\lambda x{:}B)$. (Because PHLIQA1 expressions may be denotation-less, under certain interpretations, λ-abstraction results in a *partial* function).
2. If *F* has type $(\alpha \rightarrow \beta)$ or $(\alpha \dashrightarrow \beta)$ and *E* has any type ε, then **application** *(fun: F, arg: E)* has type β. The expression may be abbreviated as *F (E)*.

Secondly, we have quantification and similar operations. They do not use variables in an explicit way: the only branching category which introduces variables is the λ-abstraction.

3. If *A* has type S (ε) or B(ε), *P* has type $(\alpha \rightarrow$ *truthvalue*$)$ or $(\alpha \dashrightarrow$ *truthvalue*$)$, and *F* has type $(\alpha \rightarrow \beta)$ or $(\alpha \dashrightarrow \beta)$, then:
universal-quantification *(forall: A, holds: P)* and
existential-quantification *(forsome: A, holds: P)* have type *truthvalue*,
selection *(head: A, mod: P)* has type S (ε),
iteration *(for: A, apply: F)* has type B (β).

[1] Landsbergen and Scha (1977) actually do this.

[2] The corresponding semantic rules are given later, in section 4.

Logical operations:

4. If *P* and *Q* have type *truthvalue*, then
   non *(arg: P)*, also written as $\neg$ P,
   conj *(1: P, 2: Q)*, also written as P & Q,
   disj *(1: P, 2: Q)*, also written as P $\vee$ Q,
   have type *truthvalue*.

Some operations which are useful to form expressions which denote the domains of compound types (see Chapter V, section 3):

5. If *A* has type S (ε) then **power** *(arg: A)* has type S (S(ε)), **bags** *(arg: A)* has type S (B(ε)), **lists** *(arg: A)* has type S (L(ε)), **files** *(arg: A)* has type S (F(ε)).
6. If A_1 has type S (ε_1), ..., A_n has type S (ε_n), then:
   cartesian-product *(1: A_1, ..., n: A_n)*, also written as $A_1 \times \ldots \times A_n$, has type S ($\langle\varepsilon_1, \ldots, \varepsilon_n\rangle$), and **union** *(1: A_1, ..., n: A_n)*, also written as $\cup$ *(A_1, ..., A_n)*, has type S ($\cup$ (ε_1, ..., ε_n)).
7. If *D* has type S (α) and *R* has type S (β), then
   functions$_t$ *(domain: D, range: R)* has type S (($\alpha \rightarrow \beta$)) and
   functions$_p$ *(domain: D, range: R)* has type S (($\alpha \dashrightarrow \beta$)).

Next we mention various other operations.

8. If *A* has type S (α) then **Count** *(arg: A)* has type *integer*.
9. If *D* has type S (α) and *E* is an expression, then **elt-of** *(E, D)* has type *truthvalue*. It is also written as $E \in D$.
10. If *A* has type S (α) then **unel** *(arg: A)* has type α.
11. If *T* has type $\langle\varepsilon_1, \ldots, \varepsilon_n\rangle$ then, for any positive integer $i \leqslant n$:
    el$_i$ *(arg: T)*, also written as *T [i]*, has type ε_i.
12. If F_1 has type $(\alpha_1 \rightarrow \beta_1)$ or $(\alpha_1 \dashrightarrow \beta_1)$, ..., F_N has type $(\alpha_n \rightarrow \beta_n)$ or $(\alpha_n \dashrightarrow \beta_n)$, then **function-choice** *(1: F_1, ..., n: F_n)* has type ($\cup$ (α_1, ..., α_n) $\rightarrow$ $\cup$(β_1, ..., β_n)).
13. If E_1 and E_2 are expressions, **equal** *(1: E_1, 2: E_2)* has type *truthvalue*. It is also written as $E_1 = E_2$.
14. If *N* has a type α such that COMPONENTS (α) $\subseteq$ {**integer, real**} and *E* has any type ε, then **amount$_u$** *(num: N, unit: E)* has type AMT (ε).
15. If *N* has a type α such that COMPONENTS (α) $\subseteq$ {**integer, real**} and *A* has a type of the form AMT (γ), then **amount$_a$** *(num: N, amount: A)* has type AMT (γ).
16. If *B* has type S ($\langle\alpha, \beta\rangle$) then **function** *(pairs: B)* has type ($\alpha \dashrightarrow \beta$).
17. If *A* has type B (α) then **bag-to-set** *(arg: A)* has type S (α).

18. If $E_1, \ldots, E_n$ have types $\varepsilon_1, \ldots, \varepsilon_n$, respectively, then *tuple$_n$ (1: E_1, ..., n: E_n)* also written as $\langle E_1, \ldots, E_n \rangle$ has type $\langle \varepsilon_1, \ldots, \varepsilon_n \rangle$
19. If for some *n NT* has type $\langle \varepsilon_1, \ldots, \varepsilon_n \rangle$ then **bag** *(tuple: NT)* has type $B(\cup(\varepsilon_1, \ldots, \varepsilon_n))$, **set** *(tuple: NT)* has type $S(\cup\varepsilon_1, \ldots, \varepsilon_n))$, **list** *(tuple: NT)* has type $L(\cup(\varepsilon_1, \ldots, \varepsilon_n))$, **file** *(tuple: NT)* has type $F(\cup(\varepsilon_1, \ldots, \varepsilon_n))$.
20. If for some *n TT* has type $\langle\langle\alpha_1, \beta_1\rangle, \ldots, \langle\alpha_n, \beta_n\rangle\rangle$ then **function** *(tuple: TT)* has type $(\cup(\alpha_1, \ldots, \alpha_n) \dashrightarrow \cup(\beta_1, \ldots, \beta_n))$.
21. If *TV* has type **truthvalue** and *E* has type ε then **cond** *(if: TV, then: E)* has type ε.
22. If *E* has type ε and *i* is an integer, then **id$_i$** *(arg: E)* has type $ID_i(\varepsilon)$.
23. If for some integer *i* and some type γ, *A* has the type $ID_i(\gamma)$, then **rid** *(arg: A)* has type γ.
24. If *P* has type $(\alpha \longrightarrow$ *truthvalue*$)$ or $(\alpha \dashrightarrow$ *truthvalue*$)$ and *E* has any type ε, then **presup** *(presup: P, descr: E)* has type ε.

If an expression is meant to be read by humans, we may use abbreviated notations instead of the "official" ones. Some abbreviations were already introduced above. Some others:

- If the selectors identify the branching category, the branching category may be left out. (For instance *(forall: S, holds: P)*, instead of: **universal-quantification** *(forall: S, holds: P).)*
- The selectors *1-n* may be left out.
- If a branching category has only one selector, the selector may be left out.

The rules 1-24 above define the type of any complex expression in terms of the types of its immediate sub-expressions; so eventually the type of any expression is defined in terms of the types of the terminals.

For any specific language, the types of the terminals are also given. We may therefore assume the existence of a function TYPE, applicable to any legitimate expression of a PHLIQA1 language, and delivering the type of that expression.

The occurrence of the variable *x* as the λ-variable in an expression of the form $(\lambda x: A)$ is called the *defining* occurrence of *x*. The expression *A* is the *scope* of this occurrence. If an occurrence of a variable is not a defining one and it is not within the scope of a defining occurrence of the same variable, the occurence is called *free*. A *closed* expression is defined as an expression that does not contain free occurrences of variables.

4. The Semantics of the PHLIQA1 Languages.

Because the languages are many-sorted, assigning an interpretation to a language consists of two steps:

1. To every atomic type, a set of entities (a *domain*) is assigned. By virtue of

the semantic rules of the type system (see below), this defines the domain of *any* type.

2. To every term, a *denotation* is assigned; this denotation must be an element of the domain of the type of the term. By virtue of the semantic rules of the language, this defines the denotation of any expression.

The semantic rules assume the following distinct primitive objects: TRUE and FALSE, the integers, the reals, the alphanumeric strings, ID and AMOUNT. Sets, bags, files, lists, functions and n-tuples are assumed to be *distinct* kinds of mathematical entities. (E.g.: a function can never be equal to a set of pairs).

The semantic rules of the type system.

A type interpretation I_{atom} assigns domains to the atomic types. I_{atom} must fulfill the following conditions:

a. for any atomic type α, $I_{atom}(\alpha)$ is a set of individuals.
b. for any two distinct atomic types α and β, $I_{atom}(\alpha)$ and $I_{atom}(\beta)$ are disjoint.
c. the domain of *truthvalue* is {TRUE, FALSE};
   the domain of *integer* is the set of integers;
   the domain of *real* is the set of reals;
   the domain of *string* is the set of alphanumeric strings.

Now we define a function DOM which assigns a domain (a set of entities) to any type. DOM is recursively defined, by means of the following rules:

1. For any atomic type α, DOM $(\alpha) = I_{atom}(\alpha)$.
2. DOM (S(β)) is the set of all subsets of DOM(β).
   DOM(B(β)) is the set of all bags whose elements are from DOM(β).
   DOM(F(β)) is the set of all files whose elements are from DOM(β).
   DOM(L(β)) is the set of all lists whose elements are from DOM(β).
3. DOM ($\langle\alpha_1, \ldots, \alpha_n\rangle$) is the set of all n-tuples $\langle A_1, \ldots, A_n\rangle$ such that $A_1 \in$ DOM(α_1), ..., $A_n \in$ DOM (α_n).
4. DOM (($\alpha \dashrightarrow \beta$)) is the set of all partial functions from the domain of α into the domain of β.
   DOM (($\alpha \rightarrow \beta$)) is the set of all total functions from the domain of α into the domain of β.
5. DOM (AMT(α)) is the set $\{\langle$AMOUNT, $\langle x, y\rangle\rangle \mid (x \in$ DOM (*integer*) $\vee$ $x \in$ DOM (*real*)) & $y \in$ DOM (α)$\}$
6. DOM ($ID_i(\alpha)$) is the set $\{\langle\langle$ID, $i\rangle, y\rangle \mid y \in$ DOM (α)$\}$
7. DOM ($\cup(\alpha_1, \ldots, \alpha_n)$) is the union of DOM ($\alpha_1$), ..., DOM ($\alpha_n$).

The semantic rules of the language.

Let a type interpretation I_{atom} be given, then a term interpretation I_{term} can be specified which assigns a denotation to each of the constants and variables of the language. I_{term} must fulfill the following conditions:
a. the denotation of every term is an element of the domain of its type,
b. the formal constants have their usual standard denotations,
c. for any atomic type α, the denotation of GS_α is the domain of α.
(Note that an interpretation also assigns denotations to variables – we do not use a separate value-assignment function for the variables. The semantic rules are such that the denotation of a closed expression does not depend on the assignment of denotations to variables.)

We now give a recursive definition of the denotation D[E] of any expression E.[3] If E is a term, $D[E] = I_{term}(E)$. If E is a complex expression **b** *(sel$_1$: E$_1$, ..., sel$_n$: E$_n$)*, its denotation is defined as follows:
- If E_1, E_2, ..., or E_n does not have a denotation, E does not have a denotation.
- If E_1, ..., and E_n all have a denotation, $D[E]$ is defined by the following rules[4]:

1. $D[(\lambda x: A)]$ is the partial function that assigns to any element e in the domain of the type of x the denotation (if there is any) that A has for the term-interpretation I'_{term} which only differs from I_{term} in that x denotes e.
2. D[**application** *(fun: F, arg: A)*] is the result of applying $D[F]$ to $D[A]$. If $D[A]$ is not an element of the domain of $D[F]$, the expression '**application** *(fun: F, arg: A)*' has no denotation.
3. D[**universal-quantification** *(forall: S, holds: P)*] is TRUE if the application of $D[P]$ to all elements of $D[S]$ yields TRUE, and FALSE otherwise.
   D[**existential-quantification** *(forsome: S, holds: P)*] is TRUE if the application of $D[P]$ to some element of $D[S]$ yields TRUE, and FALSE otherwise.
   D[**selection** *(head: S, mod: P)*] is the subset of those elements of $D[S]$ for which $D[P]$ yields TRUE.
   D[**iteration** *(for: S, apply: F)*] is the bag of all entities which $D[F]$ yields when applied to each element of $D[S]$.
4. $D[\neg TV]$ is TRUE if $D[TV]$ is FALSE; otherwise it is FALSE.
   D[**conj** *(1: A$_1$, 2: A$_2$)*] is TRUE if both $D[A_1]$ and $D[A_2]$ are TRUE; otherwise it is FALSE.
   D[**disj** *(1: A$_1$, 2: A$_2$)*] is FALSE if neither $D[A_1]$ nor $D[A_2]$ are TRUE; otherwise it is TRUE.

[3] *A, B, E, F, N, P, S, TV, NT, TT* and their indexed variants, are meta-variables which stand for PHLIQA1-expressions.

[4] These rules run parallel to the syntax rules in section 3.

5. D[**power** *(arg: S)*] is the set of subsets of $D[S]$.
   D[**bags** *(arg: A)*] is the set of all bags whose elements are from $D[A]$.
   D[**lists** *(arg: A)*] is the set of all lists whose elements are from $D[A]$.
   D[**files** *(arg: A)*] is the set of all files whose elements are from $D[A]$.
6. D[**cartesian-product** *(1: S_1, ..., n: S_n)*] is the set of all n-tuples whose first element belongs to $D[S_1]$, ..., and whose n^{th} element belongs to $D[S_n]$;
   D[**union** *(1: S_1, ..., n: S)*] is the union of the sets $D[S_1]$, ..., $D[S_n]$.
7. D[**functions$_t$** *(domain: A, range: B)*] is the set of total functions which map $D[A]$ into $D[B]$.
   D[**functions$_p$** *(domain: A, range: B)*] is the set of functions which map a subset of $D[A]$ into $D[B]$.
8. D[**Count** *(arg: S)*] is the cardinality of $D[S]$.
9. $D[E \in A]$ is TRUE if $D[E]$ is an element of $D[A]$ and FALSE if D [E] is not an element of $D[A]$.
10. If $D[S]$ is a one-element set, D[*unel (set: S)*] is its element; otherwise, *'unel (set: S)'* does not have a denotation.
11. D[**el$_i$** *(arg: NT)*] is the i-th element of the n-tuple $D[NT]$.
12. D[**function-choice** *(1: F_1, ..., n: F_n)*] is the function which yields for any argument the result of applying any applicable one from $D[F_1]$, ..., $D[F_n]$. If there are any arguments to which different functions from $D[F_n]$, ..., $D[F_n]$ are applicable and for which they yield different values then *'***function-choice** *(1: F_1, ..., n: F_n)'* has no denotation.
13. $D[A = B]$ is TRUE if $D[A]$ and $D[B]$ are identical entities; otherwise it is FALSE.
14. D[**amount$_u$** *(num: N, unit: E)*] is ⟨AMOUNT, ⟨A, B⟩⟩, where A = $D[N]$ and B = $D[E]$.
15. If $D[A]$ = ⟨AMOUNT, ⟨M, U⟩⟩ then D[**amount$_a$** *(num: N, amount: A)*] is ⟨AMOUNT, ⟨B, U⟩⟩, where B = M * $D[N]$.
16. If $D[B]$ is a set of pairs such that
    $\forall a, b, c, d: (\langle a, b\rangle \in D[B] \ \& \ \langle c, d\rangle \in D[B]) \supset (a = c \supset b = d)$
    then D[**function** *(pairs: B)*] is the function whose extension is defined by $D[B]$; otherwise, *'***function** *(pairs: B)'* has no denotation.
17. If $D[A]$ is a bag containing no duplicates, then D[**bag-to-set** *(arg: A)*] is the set containing precisely the same elements as $D[A]$; otherwise *'***bag-to-set** *(arg: A)'* has no denotation.
18. D[**tuple$_n$** *(1: E_1, ..., m: E_n)*] is the n-tuple having $D[E_1]$ as its first element, ..., $D[E_n]$ as its n^{th} element.
19. D[**bag** *(tuple: NT)*] is the bag containing all elements of $D[NT]$.
    D[**set** *(tuple: NT)*] is the set containing all elements of $D[NT]$.
    D[**list** *(tuple: NT)*] is the list containing all elements of $D[NT]$, in the same order.

***D*[file** *(tuple: NT)*] is the file containing all elements of ***D*[*NT*]**, in the order of their first occurrence.

20. ***D*[function** *(tuple: TT)*] is the function whose extension is defined by ***D*[*TT*]**. If ***D*[*TT*]** does not define a function-extension '**function** *(tuple: TT)*' has no denotation.
21. ***D*[cond** *(if: P, then: E)*] is ***D*[*E*]** if ***D*[*P*]** is TRUE; otherwise it does not have a denotation.
22. If *i* is an integer, then ***D*[$\mathbf{id}_i$** *(arg: E)*] is ⟨⟨ID, *i*⟩, ***D*[*E*]**⟩.
23. If *E* denotes, for some *i* and some A, ⟨⟨ID, *i*⟩, A⟩ then ***D*[rid** *(arg: E)*] is A.
24. ***D*[*(presup: P, descr: E)*]** is ***D*[*E*]**. (This shows that the *presup-descr* branching is semantically superfluous. The role of this branching in the language is described in section Bronnenberg et al. (1980), section 5.5.)

It must be noted that the semantic definition of every branching category is in accordance with its type definition:

If $\boldsymbol{D}[A_1] \in \text{DOM}(\text{TYPE}[A_1]) \,\&\, \ldots\ldots \,\&\, \boldsymbol{D}[A_n] \in \text{DOM}(\text{TYPE}[A_n])$
then $\boldsymbol{D}[\mathbf{b}(sel_1\text{: } A_1, \ldots, sel_n\text{: } A_n)] \in \text{DOM}(\text{TYPE}[\mathbf{b}(sel_1\text{: } A_1, \ldots, sel_n\text{: } A_n)])$.

This is a *requirement* which any branching category must fulfill.

5. Semantic Anomaly.

The phenomenon of 'semantic anomaly' is perhaps best known from the semantics of natural language. In the linguistic literature, one finds the observation that certain sentences, though syntactically well-formed, are neverteless 'weird', 'absurd' of 'deviant': sentences of the kind "The typewriter drinks the square root of the President of France".
In a formal language with a many-sorted type system, similar phenomena may be observed, in an even more clearcut way: an expression may be semantically anomalous in being tautologous, in being self-contradictory, in denoting the empty set under all interpretations the language (though using descriptive constants in the expression), in having no denotation under any interpretation of the language. Detecting that an expression has properties like those just mentioned, may be an important facility in a question-answering system: it may be worthwhile to be able to detect semantically anomalous representations of questions, since they may represent less plausible readings of the input question.

Procedures which detect the semantic anomaly of expressions must be able to use the appropriate information about the semantic types of the sub-expressions of an expression, since these indicate the range of their possible denotations. The type of any expression is defined recursively in terms of the types of its sub-expressions. (See section 3). To 'match' the types of a

function and its argument, or to compare the domain of a predicate with the range of the variable in a quantification, a procedure which checks the semantic well-formedness of sentences uses the "type-inclusion" relation. This is a relation between types, written as $\subseteq_T$, which has an important semantic property:
$\alpha \subseteq_T \beta$ implies [5] that for every interpretation the domain of α is a subset of the domain of β.

The type-inclusion relation is recursively defined, in terms of relation $\subseteq_{Tc}$ on "component types".
$\alpha \subseteq_T \beta =_{def}$
$\forall \varphi \in$ COMPONENTS (α): $\exists \psi \in$ COMPONENTS (β): $\varphi \subseteq_{Tc} \psi$
(For the definition of COMPONENTS, see section 2.)

$\varphi \subseteq_{Tc} \psi =_{def}$
if for some $\Omega \in \{L, B, S, F, AMT, ID_2, ID_1, \ldots\}$
φ has the form $\Omega(\varphi')$ & ψ has the form $\Omega(\psi')$
then $\varphi' \subseteq_T \psi'$

else if $\exists n$: φ has the form $\langle \varphi_1, \ldots, \varphi_n \rangle$ &
ψ has the form $\langle \psi_1, \ldots, \psi_n \rangle$
then $\forall i$: $\varphi_i \subseteq_T \psi_i$

else if ψ has the form $(\psi_a \dashrightarrow \psi_v)$
then φ has the form $(\varphi_a \longrightarrow \varphi_v)$ or $(\varphi_a \dashrightarrow \varphi_v)$
& $\varphi_a \subseteq_T \psi_a$ & $\varphi_v \subseteq_T \psi_v$ & $\psi_a \subseteq_T \varphi_a$

else if ψ has the form $(\psi_a \longrightarrow \psi_v)$
then φ has the form $(\varphi_a \longrightarrow \varphi_v)$
& $\varphi_a \subseteq_T \psi_a$ & $\psi_a \subseteq_T \varphi_a$ & $\varphi_v \subseteq_T \psi_v$

else φ and ψ are atomic types & $\varphi \equiv \psi$.

For instance, for different atomic types α_1, α_2, β_1, β_2, γ_1 and γ_2:

$\alpha_1 \subseteq_T \alpha_1$ holds,
$\cup(\alpha_1, \beta_1) \subseteq_T \alpha_1$ does not hold,
$\alpha_1 \subseteq_T \cup(\alpha_1, \beta_1)$ holds,
$\cup(\alpha_1, \beta_1) \subseteq_T \cup(\beta_1, \cup(\alpha_1))$ holds,

[5] In Landsbergen & Scha (1977) it is claimed that $\alpha \subseteq_T \beta$ iff for every interpretation the domain of α is a subset of the domain of β. But the implication from right to left is not valid – neither for the definition of $\subseteq_T$ which was used in the 1977 paper, nor for the current one.

$S(\alpha_1) \subseteq_T S(\cup(\alpha_1, \beta_1))$ holds,
$(\alpha_1 \rightarrow \beta_1) \subseteq_T (\alpha_1 \rightarrow \cup(\beta_1, \gamma_1))$ holds,
$(\alpha_1 \rightarrow \beta_1) \subseteq_T (\alpha_1 \dashrightarrow \beta_1)$ holds,
$\langle \cup(\alpha_1, \alpha_2), \cup(\beta_1, \beta_2) \rangle \subseteq_T$
$\cup(\langle \cup(\alpha_1, \alpha_2), \cup(\beta_1, \gamma_1) \rangle, \langle \alpha_1, \beta_2 \rangle, \langle \alpha_2, \cup(\beta_2, \gamma_2) \rangle, \langle \gamma_2, \beta_1 \rangle)$ holds.

Equality of types, written as "$=_T$", is defined as mutual inclusion:

$\alpha =_T \beta \quad =_{def} \quad \alpha \subseteq_T \beta \ \& \ \beta \subseteq_T \alpha.$

For instance, for different atomic types α and β:

$\alpha =_T \beta$ does not hold,
$\alpha =_T \alpha$ holds,
$S(\alpha) =_T \cup(S(\alpha))$ holds,
$\cup(S(\alpha), S(\beta)) =_T S(\cup(\alpha, \beta))$ does not hold.

Using the relation $\subseteq_T$, we can now give the following syntactic definitions of function-application, quantification, iteration and selection, which impose stricter demands on the types of their sub-expressions than the definitions given before.

If *F* has type $(\alpha \rightarrow \beta)$ or $(\alpha \dashrightarrow \beta)$, *E* has type ε
and $\varepsilon \subseteq_T \alpha$, then **application** *(fun: F, arg: E)* has type β.

If *A* has type $S(\varepsilon)$ or $B(\varepsilon)$, *P* has type $(\alpha \rightarrow$ **truthvalue**$)$ or
$(\alpha \dashrightarrow$ **truthvalue**$)$, *F* has type $(\alpha \rightarrow \beta)$ or
$(\alpha \dashrightarrow \beta)$, and $\varepsilon \subseteq_T \alpha$, then:
universal-quantification *(forall: A, holds: P)*
and
existential-quantification *(forsome: A, holds: P)*
have type *truthvalue*,
selection *(head: A, mod: P)* has type $S(\varepsilon)$,
iteration *(for: A, apply: F)* has type $B(\beta)$.

6. Additions and Abbreviations.

The previous sections of this Appendix have defined a fragment of the PHLIQA1 language which is a slight extension of the language defined in Bronnenberg et al. (1980). However, in Chapters II and V the examples lay often outside the boundaries of this fragment; they use some semantic operations which I did not include in the Bronnenberg et al. language

because they are not particularly interesting. For the sake of readability, I have also used some abbreviations which deviate considerably from the PHLIQA1 notation. I shall now list these additions and abbreviations.

Additional operations.

1. If *A* has type B*(integer)* or S*(integer)*, **Sum** *(arg: A)* has type *integer*. It denotes the sum of the elements of $\boldsymbol{D}[A]$.
2. If *N* has type *integer*, **Ints** *(arg: N)* has type S*(integer)*. It denotes the set of integers *i* such that $0<i\leqslant \boldsymbol{D}[N]$.
3. If *M* and *N* have type *integer* than **greater-than** *(1:M, 2:N)*, also written as $M>N$, and **smaller-than** *(1:M, 2:N)*, also written as $M<N$, have type *truthvalue*, and have their usual meaning.
4. If *A* and *B* have type $S(\alpha)$ and $S(\beta)$, than **intersection** *(1:A, 2:B)* has type $S(\cup(\gamma_1, \ldots, \gamma_n))$, where $\{\gamma_1, \ldots, \gamma_n\}$ = COMPONENTS$(\alpha) \cap$ COMPONENTS(β); it has the obvious meaning, and may also be written as $A \cap B$.
5. If *A* has a type of the form $B(S(\alpha))$ or $S(S(\alpha))$, then **union** *(arg: A)*, also written as $\cup$ *(A)*, has type $S(\alpha)$, with the obvious meaning.

Additional Abbreviations.

$\{A_1, \ldots, A_n\}$ stands for **set** $(\langle A_1, \ldots, A_n\rangle)$.
$(\lambda x_1, \ldots, x_1\text{: } E)$ with x_1 of type α_1, ..., x_n of type α_n, stands for $(\lambda u\text{: } E')$ where *u* has type $\langle\alpha_1, \ldots, \alpha_n\rangle$ and $E' = E[x_1 := u[1], \ldots, x_n := u[n]]$.

If *A* has type $S(\alpha)$ or $B(\alpha)$ and *B* has type *truthvalue*, then:
$\{x \in A \mid B\}$ stands for **selection** *(head: A, mod: (λx: B))*,
$[\iota x \in A \mid B]$ stands for **unel** *(arg:* **selection** *(head: A, mod: (λx: B)))*,
$(\forall x \in A\text{: } B)$ stands for **universal-quantification** *(forall: A, holds: (λx:* **B**)),
$(\exists x \in A\text{: } B)$ stands for
existential-quantification *(forsome: A, holds: (λx: B))*,
$(\exists_n x \in A\text{: } B)$ stands for **Count** *(arg:* **selection** *(head: A, mod: (λx: B)))* $= n$,
$\boldsymbol{P}(A)$ stands for **power** *(arg: A)*,
$\boldsymbol{P}_n(A)$ stands for **selection** *(head:* **power** *(arg: A), mod: (λx:* **Count**$(x) = n)$),
where *x* has type α.
If *F* has type $(\langle\alpha\rangle \rightarrow \beta)$ or $(\langle\alpha\rangle \dashrightarrow \beta)$ and *C* has type α, then $F(\langle C\rangle)$ may be written as $F(C)$.

In syntactic positions where expressions with a type of the form $S(\alpha)$ are required, I have also allowed expressions with a type of the form $B(\alpha)$ in some examples. If *A* is such an expression one should, in such cases, replace $\boldsymbol{D}[A]$ by $\boldsymbol{D}$[**bag-to-set** *(A)*] in the semantics.

Remark.

In the PHLIQA1 system, the PHLIQA1 languages are always used in an extensional way – a descriptive atomic type *possible-world* is nowhere assumed.

In Chapter II, where the PHLIQA1 treatment of questions and answers is being compared with some intrinsically intensional treatments of questions and answers, I have however taken the liberty to combine the notation defined above with the use of intension- and extension-operators in the style of Montague's (1973) IL. I hope the syntax and semantics of this "hybrid language" are sufficiently self-evident, so that I am justified in abstaining from an explicit definition.

Nederlandstalige Samenvatting

Dit proefschrift poogt een theoretische onderbouwing te gevan aan enkele belangrijke aspecten van zg. "vraag-antwoordsystemen" – computerprogramma's die in natuurlijke taal gestelde vragen over een bepaald onderwerpgebied kunnen beantwoorden. Bij het ontwerp van zo'n vraag-antwoordsysteem moeten impliciete of expliciete beslissingen worden genomen over enkele fundamentele problemen: hoe representeert het programma de inhoud van de vragen die het beantwoordt en van de antwoorden die het geeft; en hoe wordt de "kennis" gerepresenteerd die het programma gebruikt om de vragen te beantwoorden.

Dit proefschrift pleit ervoor, vraag-antwoordsystemen te baseren op expliciete, nauwkeurig geformuleerde beslissingen over deze problemen. Het laat zien hoe het begrippen-arsenaal van de logische model-theorie voor dit doel kan worden toegepast. Vanuit dit perspectief bespreekt het proefschrift een samenhangend geheel van ideeën over de representatie van vragen, antwoorden en kennis, dat ten grondslag ligt aan het vraag-antwoordsysteem PHLIQA1 (een programma dat ontwikkeld werd bij Philips' Natuurkundig Laboratorium te Eindhoven, en waarin de auteur een belangrijk aandeel had). Andere, reeds bestaande methodes voor de representatie van vragen, antwoorden en kennis worden eveneens vanuit dit perspectief beschouwd.

Hoofdstuk I vergelijkt de hier voorgestane benaderingswijze met mogelijke alternatieven. In dat kader worden doelstellingen en werkwijze van de Kunstmatige Intelligentie besproken en de verdiensten van de Procedurele Semantiek worden met die van de Modeltheorie vergeleken.

Hoofdstuk II bespreekt methodes voor het representeren van vragen en antwoorden. Het toont de ontoereikendheid aan van voorstellen hieromtrent die recentelijk in het kader van de theoretische linguistiek en de philosophische logica ontwikkeld zijn. Een adequater voorstel wordt in enig detail uiteengezet.

Hoofdstuk III beschrijft de structuur van het vraag-antwoordsysteem PHLIQA1. De Hoofdstukken IV en V bespreken de twee kennis-representatie-methodes waarop PHLIQA1 gebaseerd is. In Hoofdstuk IV wordt een logische analyse gegeven van de wijze waarop kennis gerepresenteerd wordt in een "gewoon" gegevensbestand (georganiseerd volgens b.v. de relationele principes of de CODASYL principes). In Hoofdstuk V wordt een kennis-representatie-methode uiteengezet die gebruik maakt van "vertaalregels". Deze methode is bijzonder geschikt om de kennis te representeren die aangeeft hoe begrippen uit de natuurlijke taal samenhangen met de begrippen in termen waarvan het gegevensbestand georganiseerd is. In Hoofdstuk VI worden varianten van deze methode aangegeven, en worden alternatieven kritisch besproken.

Hoofdstuk VII trekt algemene conclusies over de "ontwerpstijl" die in het boek gedemonstreerd wordt. Een Appendix definieert een rijke logische taal, die op diverse punten in het boek in illustratieve voorbeelden gebruikt wordt.

STELLINGEN

behorende bij het proefschrift *Logical Foundations for Question Answering* van Remko J.H. Scha.

Groningen, 17 februari 1983

1. Verbale interacties hebben een expliciet gemarkeerde syntactisch/semantische structuur die beschreven kan worden als een aaneenrijging en recursieve nesting van „discourse-eenheden". De bezwaren tegen de recursieve structuur aangevoerd door Bruce en Reichman zijn niet steekhoudend.

   B. Bruce: Discourse Models and Language Comprehension.
   American Journal of Computational Linguistics, 1975, Microfiche-35.

   R. Reichman: *Plain Speaking: A Theory and Grammar of Spontaneous Discourse*. BBN Technical Report nr. 4681.
   Cambridge, Mass.: Bolt, Beranek and Newman, Inc., 1981.

   L. Polanyi and R.J.H Scha: On the Recursive Structure of Discourse.
   Proceedings of the Tilburg Symposium on Connectedness in Sentence, Text and Discourse, January, 1982. Universiteit van Tilburg.

   L. Polanyi and R.J.H. Scha: The Syntax of Discourse. Te verschijnen in: *TEXT*.

2. Analyse van de protocollen van de vaak opmerkelijke interacties tussen menselijke personen en primitieve dialoogsystemen als ELIZA en PARRY zou een nuttige uitbreiding vormen van het repertoire van Garfinkeliaanse experimentele technieken voor onderzoek naar de interactieve constructie van sociale realiteit.

   J. Weizenbaum: ELIZA – A Computer Program for the Study of Natural Language Communication Between Man and Machine. *Comm. ACM* **9**, 1966, 36-45.

   K.M. Colby, J.B. Watt and J.P. Gilbert: A Computer Method of Psychotherapy: Preliminary Communication.
   J. Nerv. Mental Disease **142**, 1966, 148-152.

   H. Garfinkel: Studies of the Routine Grounds of Everyday Activities. *Social Problems* **11**, 1964, 225-250.

3. Gedurende het verloop van een narratieve tekst bepaalt het aspekt van de hoofdzinnen of het referentiepunt voor de temporele interpretatie van de uitingen wel of niet verschuift.

   L. Polanyi and R.J.H. Scha: Towards a formal semantics of stories in conversation (Abstract). *Colloquium on discourse representation*, Kleve, September 15-18, 1981.

   E. Hinrichs: Temporale Anaphora im Englischen. Ongepubliceerd Manuscript, Universiteit van Tübingen, 1981.

4. Er bestaat geen syntactisch onderscheid tussen collectieve en distributieve werkwoorden.

   R.J.H. Scha: Distributive, collective and cumulative quantification. In: J.A.G. Groenendijk, T.M.V. Janssen en M.B.J. Stokhof (eds.): *Formal Methods in the Study of Language*, Vol. 2. Amsterdam: Mathematisch Centrum, 1981.

5. Zich vertakkende quantificatiestructuren spelen geen rol in de semantiek van natuurlijke taal.

   J. Hintikka: Quantifiers vs. Quantification Theory. *Linguistic Inquiry* **5**, 1974, 153-177.

   G. Fauconnier: Do Quantifiers Branch? *Linguistic Inquiry* **6**, 1975, 555-578.

   J. Barwise: On Branching Quantifiers in English. *Journal of Philosophical Logic* **8**, 1979, 47-80.

6. Extensionele adjectieven kunnen niet geconjugeerd worden met intensionele. Montague's behandeling van extensionele adjectieven is daarom onjuist.

   R. Montague: The Proper Treatment of Quantification in Ordinary English. In: J. Hintikka, J. Moravcsik and P. Suppes (eds.): *Approaches to Natural Language. Proceedings of the 1970 Workshop on Grammar and Semantics.* Dordrecht: Reidel, 1973, 221-242.

7. Postulaten die tot doel hebben om de betekenis vast te leggen van intuitief ongeïnterpreteerde ,,semantische tussenstappen'' (zoals Montague's pseudo-intensionele werkwoorden en adjectieven, en de pseudo-collectieve werkwoordslezingen die ik voorgesteld heb i.v.m. stelling 4) vervullen dit doel alleen wanneer zij de elimineerbaarheid van deze ,,hulp-eenheden'' garanderen. Het verdient daarom de voorkeur zulke postulaten niet als axioma's te formuleren maar als locale vertaalregels.

   Dit proefschrift, Hoofdstuk V.

   B. Indurkhya: *Sentence Analysis Programs Based on Montague Grammar.* Eindhoven: Philips International Institute of Technological Studies. 1981.

8. LISP is niet geschikter voor het schrijven van gecompliceerde ,,intelligente'' programma's dan de meeste recentere talen uit de ALGOL-familie. De speciaal in de context van de Kunstmatige Intelligentie ontwikkelde talen met ,,ingebouwde deductie'' e.d. zijn *minder* geschikt, omdat ze de programmeur de verantwoordelijkheid ontnemen voor de algorithmische structuur van zijn programma.

9. De Kunstmatige Intelligentie produceert ,,beelden'' van de uiterlijke verschijningsvorm van de menselijke cognitie - geen ,,modellen'' van de interne structuur ervan. Het is dus eerder een artistieke discipline dan een empirische wetenschap.

   S.K. Langer: *Mind: An Essay on Human Feeling,* Vol. 1. Baltimore: Johns Hopkins University Press, 1967, 59-68.

10. Een onderzoek naar de aard van het sociale instituut ,,kunst'' kan niet de plaats innemen van een onderzoek naar de aard van aesthetische processen.

    G. Dickie: *Art and the Aesthetic. An Institutional Analysis.* Ithaca: Cornell University Press, 1974.

11. Rock 'n roll is onze enige levende artistieke traditie.

12. De gedachte van een objectieve kunst die de werkelijkheid voor zichzelf laat spreken, rond 1960 veelvuldig geponeerd (Cage, De Vries, Nouveau Réalisme, Fluxus), verdient het om alsnog ernstig in praktijk gebracht te worden. Betekenisvolle beelden hoeven niet met mensenhand gemaakt te zijn.

13. De aesthetische principes van de Nederlandse Nul-groep zijn het helderst geformuleerd en het eerst verwerkelijkt door Wladyslaw Strzeminski.

    W. Strzeminski: Untitled Statements. *Abstraction Création Art Non Figuratif*, Vol. 1, 1932, 35; Vol. 2, 1933, 40.

    H. Peeters: 0 = nul. De Nieuwe Tendenzen. *Museumjournaal* **9**, 1964, 134-145.

14. Mandelbrot's voorstellen voor de definitie van de vorm van de loop van rivieren zijn uiterst onbevredigend. Een serieuzer voorstel kan worden ontwikkeld door de Mandelbrot/Lévy definitie van de vorm van berglandschappen als uitgangspunt te nemen.

    B.B. Mandelbrot: *Fractals. Form, Chance and Dimension*. San Francisco: W.H. Freeman and Co., 1977.

    P. Lévy: *Processus stochastiques et mouvement brownien*. Parijs: Gauthier Villars, 1948.

15. Aan muzikale noties als rhythme, swing en harmonie liggen mechanische verschijnselen ten grondslag.

    Remko Scha: *Machine Guitars*. KR 006. Eindhoven: Kremlin Products, 1982.

    Remko Scha: *Guitar Mural 1*. Groningen: Taal Beeld Geluid, 1982.