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 occurrence 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 philosophical 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) 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).
question in English

ENGLISH-to-EFL translation

expression of English-oriented Formal Language

EFL-to-WML translation

expression of World Model Language

WML-to-DBL translation

expression of Data Base Language

Evaluation and answer formulation

Data Base

answer

Fig. 2. A global diagram of the PHLIQAI 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 super-language 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 \( EI \):

\[
\forall x \in EI(A): \exists y \in EI(B): x \equiv y \quad \& \quad \forall y \in EI(B): \exists x \in EI(A): x \equiv y.
\]

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 PHILIQA1 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

\[
\langle entity, entity \rangle \rightarrow truthvalue,
\]

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

\[
\langle entity, ..., entity \rangle \rightarrow truthvalue,
\]

depening on the number of arguments that the verb takes.

For instance:
- for "to exist", EXIST with type $\langle entity \rangle \rightarrow truthvalue$,
- for the main verb "to be", BE with type $\langle entity, entity \rangle \rightarrow truthvalue$,
- for the main verb "to have", HAVE with type $\langle entity, entity \rangle \rightarrow truthvalue$,
- for "to possess", POSSESS with type $\langle entity, entity \rangle \rightarrow truthvalue$.

For every adjective there is a constant of type

\[
entity \rightarrow truthvalue,
\]

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 parsimonious 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 calendar 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 *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 $\alpha$ there is a constant of type $S(\alpha)$ which denotes, under every interpretation, the domain of $\alpha$. This constant is written as: $GS_{\alpha}$. For instance: $GS_{company}$, $GS_{site}$, etc.

The other descriptive constants are functions:

<table>
<thead>
<tr>
<th>Function</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>F-CY-NAME</td>
<td>$(company \rightarrow string)$</td>
</tr>
<tr>
<td>F-COUNTRY-NAME</td>
<td>$(country \rightarrow string)$</td>
</tr>
<tr>
<td>F-CITY-NAME</td>
<td>$(city \rightarrow string)$</td>
</tr>
<tr>
<td>F-CPUMODEL-NAME</td>
<td>$(cpumodel \rightarrow string)$</td>
</tr>
<tr>
<td>F-PERMODEL-NAME</td>
<td>$(per-model \rightarrow string)$</td>
</tr>
<tr>
<td>F-PERTYPE-NAME</td>
<td>$(per-type \rightarrow string)$</td>
</tr>
<tr>
<td>F-CALMONTH-NAME</td>
<td>$(calmonth \rightarrow string)$</td>
</tr>
<tr>
<td>F-SITE-ADDRESS</td>
<td>$(site \rightarrow string)$</td>
</tr>
<tr>
<td>F-SITE-CITY</td>
<td>$(site \rightarrow city)$</td>
</tr>
<tr>
<td>F-SITE-COMPANY</td>
<td>$(site \rightarrow company)$</td>
</tr>
<tr>
<td>F-CITY-COUNTRY</td>
<td>$(city \rightarrow country)$</td>
</tr>
<tr>
<td>F-CPUMODEL-SITE</td>
<td>$(cpumodel \rightarrow site)$</td>
</tr>
<tr>
<td>F-PERMODEL-SITE</td>
<td>$(per-model \rightarrow site)$</td>
</tr>
<tr>
<td>F-CONF-SITE</td>
<td>$(conf \rightarrow site)$</td>
</tr>
<tr>
<td>F-CMEM-SIZE</td>
<td>$(cmem \rightarrow AMT (mem-unit))$</td>
</tr>
<tr>
<td>F-CMEM-CONF</td>
<td>$(cmem \rightarrow conf)$</td>
</tr>
<tr>
<td>F-CPU-CONF</td>
<td>$(cpu \rightarrow conf)$</td>
</tr>
<tr>
<td>F-PERIPH-CONF</td>
<td>$(periph \rightarrow conf)$</td>
</tr>
<tr>
<td>F-CPU-MONTH-INST</td>
<td>$(cpu \rightarrow calmonth)$</td>
</tr>
<tr>
<td>F-CPU-YEAR-INST</td>
<td>$(cpu \rightarrow calyear)$</td>
</tr>
<tr>
<td>F-PERIPH-MONTH-INST</td>
<td>$(periph \rightarrow calmonth)$</td>
</tr>
<tr>
<td>F-PERIPH-YEAR-INST</td>
<td>$(periph \rightarrow calyear)$</td>
</tr>
<tr>
<td>F-CPU-CPUMODEL</td>
<td>$(cpu \rightarrow cpumodel)$</td>
</tr>
<tr>
<td>F-PERIPH-PERMODEL</td>
<td>$(periph \rightarrow permodel)$</td>
</tr>
<tr>
<td>F-PERMODEL-PRICE</td>
<td>$(permodel \rightarrow AMT (money-unit))$</td>
</tr>
<tr>
<td>F-CPUMODEL-PRICE</td>
<td>$(cpumodel \rightarrow AMT (money-unit))$</td>
</tr>
<tr>
<td>F-PERMODEL-PERTYPE</td>
<td>$(permodel \rightarrow pertype)$</td>
</tr>
<tr>
<td>F-CALMONTH-NR</td>
<td>$(calmonth \rightarrow integer)$</td>
</tr>
<tr>
<td>F-CALYEAR-NR</td>
<td>$(calyear \rightarrow integer)$</td>
</tr>
<tr>
<td>F-CMEM-PRICE</td>
<td>$(cmem \rightarrow AMT (money-unit))$</td>
</tr>
</tbody>
</table>

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

5) See Chapter IV, section 4, for a more detailed discussion.
(and, as always, we have for every referential atomic type \( \tau \) a constant \( GS_\tau \), denoting its domain).

For every attribute of record type \( R \), we have a function constant of type \( (\alpha_R \rightarrow string) \) or \( (\alpha_R \rightarrow integer) \), 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:*

\[ corporation_D, site_D, configuration_D, cpu-model_D, country_D, manufacturer_D, peripheral-group_D, peripheral-model_D, peripheral-type_D. \]

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

\[ F\text{-CORP\text{\textunderscore}NAME}_D \text{ with type } (corporation_D \rightarrow string) \]
\[ F\text{-COUNTRY\textunderscore NAME}_D \text{ with type } (country_D \rightarrow string) \]
\[ F\text{-SITE\textunderscore STREETADDRES}_D \text{ with type } (site_D \rightarrow string) \]
\[ F\text{-SITE\textunderscore CITYNAME}_D \text{ with type } (site_D \rightarrow string) \]
\[ F\text{-CONF\textunderscore DATEINST}_D \text{ with type } (conf_D \rightarrow integer) \]
\[ F\text{-CONF\textunderscore CORESIZE}_D \text{ with type } (conf_D \rightarrow integer) \]
\[ F\text{-PERGROUP\textunderscore QUANTITY}_D \text{ with type } (peripheral-group_D \rightarrow integer) \]
\[ F\text{-PERMODEL\textunderscore NAME}_D \text{ with type } (peripheral-model_D \rightarrow string) \]
\[ F\text{-PERMODEL\textunderscore PRICE}_D \text{ with type } (peripheral-model_D \rightarrow integer) \]
\[ F\text{-PERTYPE\textunderscore NAME}_D \text{ with type } (peripheral-type_D \rightarrow string) \]
\[ F\text{-CPUMODEL\textunderscore NAME}_D \text{ with type } (cpu-model_D \rightarrow string) \]
\[ F\text{-CPUMODEL\textunderscore PRICE}_D \text{ with type } (cpu-model_D \rightarrow integer) \]
\[ F\text{-MANUFACTURER\textunderscore NAME}_D \text{ with type } (manufacturer_D \rightarrow string) \]
\[ F\text{-SITE\textunderscore CORP}_D \text{ with type } (site_D \rightarrow corporation_D) \]
\[ F\text{-SITE\textunderscore COUNTRY}_D \text{ with type } (site_D \rightarrow country_D) \]
\[ F\text{-CONF\textunderscore SITE}_D \text{ with type } (configuration_D \rightarrow site_D) \]
\[ F\text{-CONF\textunderscore CPUMODEL}_D \text{ with type } (configuration_D \rightarrow cpu-model_D) \]
\[ F\text{-PERGROUP\textunderscore CONF}_D \text{ with type } (peripheral-group_D \rightarrow configuration_D) \]
\[ F\text{-PERGROUP\textunderscore PERMODEL}_D \text{ with type } (peripheral-group_D \rightarrow peripheral-model_D) \]
\[ F\text{-PERMODEL\textunderscore PERTYPE}_D \text{ with type } (peripheral-model_D \rightarrow peripheral-type_D) \]
\[ F\text{-PERMODEL\textunderscore MANUF}_D \text{ with type } (peripheral-model_D \rightarrow manufacturer_D) \]
\[ F\text{-CPUMODEL\textunderscore MANUF}_D \text{ with type } (cpu-model_D \rightarrow manufacturer_D) \]
For each function $f_{\text{ALFA-BETA}}$ with a type $(\alpha \to \beta)$ where $\beta$ is not string or integer, the inverse function $f_{\text{BETA-ALFAS}}$ with type $(\beta \to S(\alpha))$ is also part of DBL.

Those functions which have a type $(\alpha \to \beta)$ where $\beta$ is a descriptive type of DBL are shown in fig. 4. The boxes represent types. An arrow pointing from a box labelled $\alpha$ to a box labelled $\beta$ represents a function constant with type $(\alpha \to \beta)$ and its inverse, with type $(\beta \to S(\alpha))$.

**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\textsuperscript{−} translation
Simplification
EFL\textsuperscript{−}-to-WML translation
Simplification
WML-to-DBL\textsuperscript{*} translation
Simplification
DBL\textsuperscript{*}-to-DBL translation
Evaluation
Answer-formulation.

− At most of the levels a simplification module is called, which converts an expression of a PHLIQA\textsubscript{1} 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\textsuperscript{−}. 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\textsuperscript{−} 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\textsuperscript{*}. DBL\textsuperscript{*} is a language which is derived from DBL: DBL\textsuperscript{*} 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\textsuperscript{*} level, because certain DBL\textsuperscript{*} 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 algorithm⁶ 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-EXRES 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),
- \(buf\text{text}\) (whose values are strings),
- \(buf\text{level}\) (whose values are integers), and
- \(false\text{-presups}\) (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
  buf\text{level} := 0; buf\text{text} := "your question is considered ungrammatical";
  for X_1 through ENGLISH-TO-EFL (READ) do
    (if buf\text{level} < 1 then (buf\text{level} := 1;
                          buf\text{text} := "your question is meaningless in the subject domain");

    for X_2 through EFL-TO-EFL (X_1)
      (do (X_3: = SIMPLIFY (WML-TO-DBL (SIMPLIFY (EFL-TO-WML (SIMPLIFY(X_2)))));
         if ASTERISK (X_3) then (if buf\text{level} < 2 then (buf\text{level} := 2;
                          buf\text{text} := "the database does not contain the information which
                          is needed to answer your question")
         else (false-presups := FALSE;
               for X_3 through PRESUP-EXRES (X_3) do
                  (if EVAL (X_3) = FALSE then false-presups := TRUE);
               if false-presups then (if buf\text{level} < 3 then (buf\text{level} := 3;
                                                      buf\text{text} := "your question contains a false presupposition")
               else (PRINT(ANSWER(EVAL(X_3)));
                        buf\text{level} := 4; buf\text{text} := "");
                  PRINT ("Do you want a search for another interpretation of your
                          question?";)
                      if READ = "no" then exit));

  PRINT (buf\text{text})
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.