American Journal of Computational Linguistics *Microfiche 32 : 72*

P H L I Q A 1 : MULTILEVEL SEMANTICS IN QUESTION ANSWERING

P. MEDEMA, W. J. BRONNENBERG, H. C. BUNT, S. P. J. LANDSBERGEN,
R. J. H. SCHA, W. J. SCHOENMAKERS, AND E. P. C. VAN UTTEREN

Philips Research Laboratories
Eindhoven, The Netherlands

ABSTRACT

This paper outlines a recently implemented question answering system , called PHLIQA 1 , which answers English questions about a data base .

Unlike other existing systems , that directly translate a syntactic deep structure into a program to be executed , PHLIQA 1 leads a question through several intermediate stages of semantic analysis . In every stage the question is represented as an expression of a formal language. The paper describes some features of the languages that are successively used during the analysis process : the English–oriented Formal Language , the World Model Language and the Data Base Language . Next , we show the separate conversion steps that can be distinguished in the process. We indicate the problems that are handled by these conversions , and that are often neglected in other systems .

1. Introduction

PHLIQA 1 is an experimental system for answering isolated English questions about a data base . We have singled this out as the central problem of question answering , and therefore postponed the treatment of declaratives and imperatives , as well as the analysis of discourse until a later version of the system . The data base is about computer installations in Europe and their users . At the moment , it is small and resides in core – but its structure and content are those of a realistic Codasyl format data base on disk (CODASYL Data Base Task Group [1971]) .

Only one module of the system , the "evaluation component" , would have to be changed in order to handle a "real" data base .

2. PHLIQA 1 's top level design

Like other recent QA systems (e.g. Petrick [1973] , Plath [1973] , Winograd [1972] , Woods [1972]) , the PHLIQA 1 system can , on the most global level , be divided into 3 parts (see fig. 1) :

– Understanding the question : Translating the question into a formal expression which represents its meaning with respect to the world model of the system.

– Computing the answer : Elaborating this expression , thereby finding the answer, as it is represented in the system' s internal formalism.

– Formulating the answer : Translating this answer into a form that can be more readily understood .

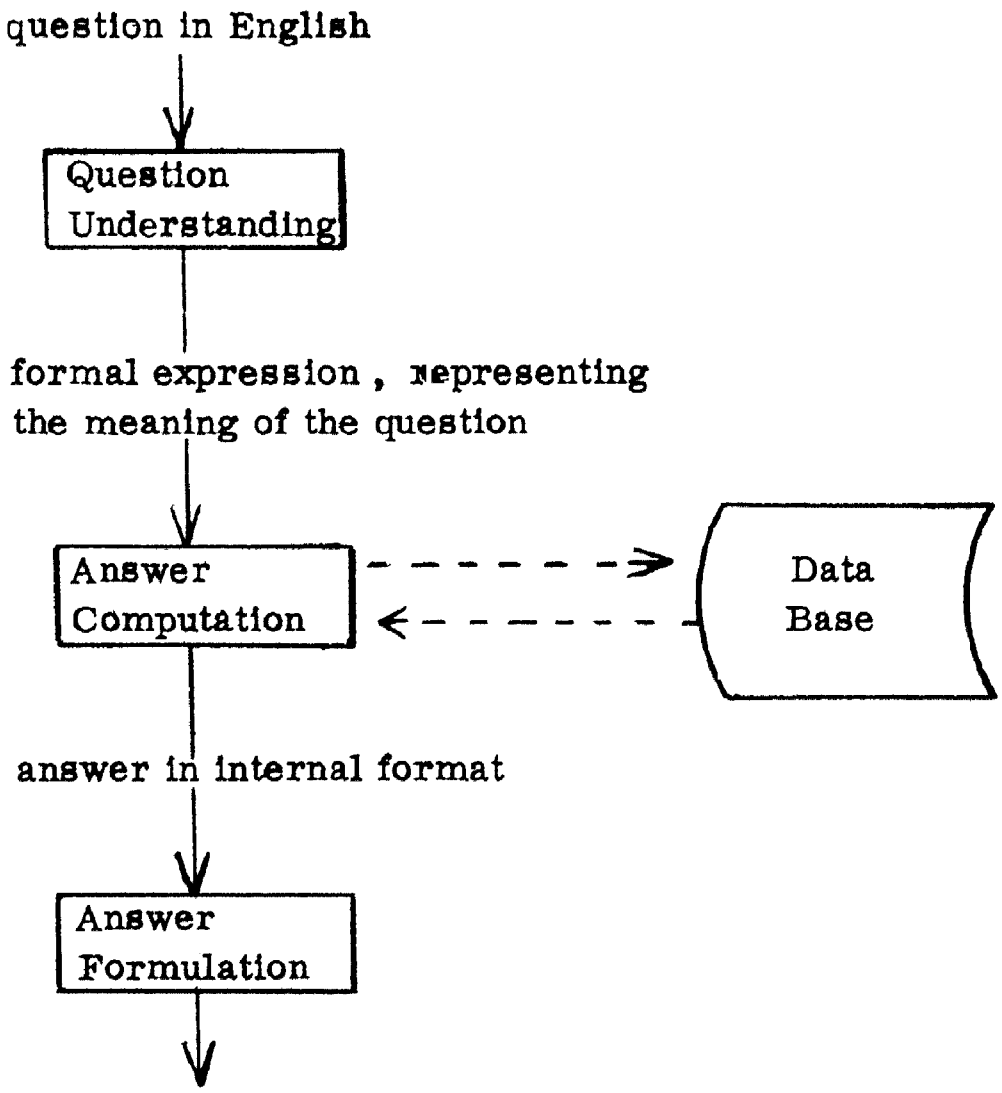


Fig . 1. Global subdivision of PHLIQA 1.

The interface between the Question Understanding component and the Answer Computation component is a formal language , called the World Model Language (WML) . Expressions of this language represent the meaning of questions with respect to the world model of the system. Its constants correspond to the concepts that constitute the universe of discourse . The language is independent of the input language that is used (in this case English) , and also independent of the storage structure of the data base .

If we now look at a further subdivision of the components , the difference between PHLIQA 1 and other systems becomes apparent . Both above and below the World Model level , there is an intermediate stage of analysis , characterized by a formal language , resp :

- The English–oriented Formal Language (EFL) , which contains constants that correspond to the terms of English . This language is used to represent the " semantic deep structure " of the question . That divides the Question Understanding component into two successive subcomponents :
  a. Constructing an EFL expression , using only linguistic knowledge .
  b. Translating the EFL expression into a WML expression , by taking knowledge about the structure of the world into account .
- The Data Base Language (DBL) , which contains constants that correspond to data base primitives . (The World Model constants do not correspond to data base primitives , because we want to handle a " realistic " data base : one that was designed to be stored efficiently , rather than to reflect neatly the structure of the world .)
  This splits the Answer Computation component into two successive subcomponents :
  a. Translating a WML expression into a DBL expression taking knowledge about the data base structure into account.
  b. Evaluating the DBL expression .

The set–up of the system that one arrives at in this way , is shown in fig. 2. In section 3 , we say something more about PHLIQA' s formal languages in general . How the three successive translation modules are further divided into smaller modules , called "convertors" , is discussed in the sections 4 , 5 and 6. Section 7 treats the evaluation component . The Answer Formulation component is very primitive , and will not be considered further .

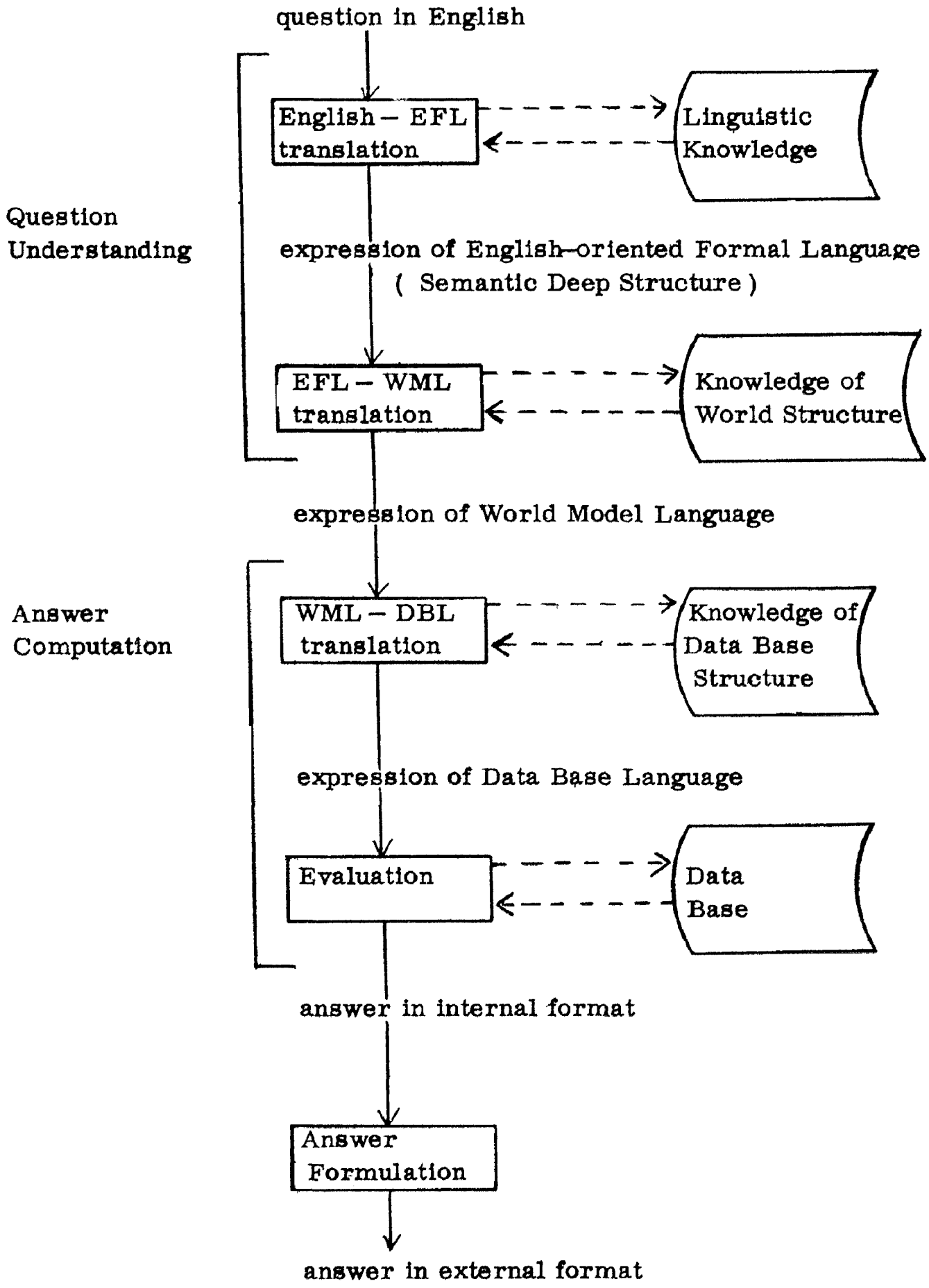


Fig. 2. PHLIQA 1 's main components .

3. PHLIQA 1's formal languages

3.1. Syntax

The three PHLIQA languages (the English-oriented Formal Language , the World Model Language and the Data Base Language) have largely identical syntactic definitions . As pointed out already , their most important difference is in the constants they contain . They share most , but not all , syntactic constructions .

PHLIQA expressions are " trees " that consists of terminal nodes (constants and variables) and syntactic constructions . A syntactic construction is an unordered collection of labeled branches , departing from one node .

The branches of a PHLIQA " tree " can converge to a common subtree .

Using a system of semantic types , the syntax of a PHLIQA language defines how expressions can be combined to form a larger expression . For every syntactic construction , there is a rule which specifies :

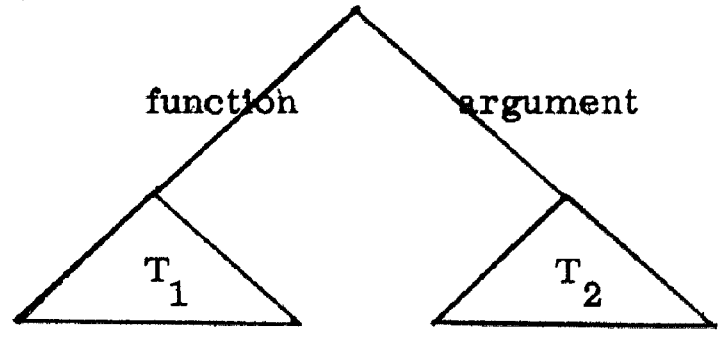
- What the semantic types of its immediate sub-expressions are allowed to be .
  (There is never a restriction on the syntactic form of the sub-expressions .)
- How the semantic type of the resulting expression is derived from the semantic types of the immediate sub-expressions .

Given the types of the elementary expressions (the constants and variables) , this defines the language . (Sources of inspiration for the syntax of our formal languages were the Vienna Definition Language (Wegner [1972]) , and a formulation of Higher – Order Logic by J.A. Robinson [1969] .)

Some simple examples of semantic types are the following :

A constant representing a single object has a simple type . E.g. , " 6 " has the type " integer " . A constant representing a collection of objects of type α has a type of the form $\langle\alpha\rangle$. E.g. , " companies " has the type "<company>", " integers " has the type "<integer>" .

A constant representing a function that can have arguments of type α and values of type β has the type $\alpha \rightarrow \beta$. E.g. , the function " f–company–sites " has the type " company $\rightarrow$ ⟨site⟩", the function " f–sum " has the type " ⟨integer⟩ $\rightarrow$ integer " .
The syntactic rule for the construction " function – application " could state that the expression



is well – formed if T_2 is a well–formed expression of type α and T_1 is a well – formed expression of type $\alpha \rightarrow \beta$, where α and β may be any type ; the whole expression then has the type β
The PHLIQA languages contain a wide variety of syntactic constructions , e.g. constructions for different kinds of quantification , for selecting elements from a list , for reordering a list , etc .

3. 2. Semantics

The PHLIQA languages have a formal semantics which recursively defines the values of the expressions. This definition assumes as primitive notions the denotations of the constants of the language : function – constants denote procedures , and the other constants denote value – expressions . This means that if we know the denotations of the constants occurring in an expression , the value of the expression is defined by the semantic rules of the language . For the Data Base Language , we indeed know the denotations of the constants ; what we call the data base is nothing but the implementation of the " primitive procedures ", i.e. : the procedures corresponding to DBL functions , and the procedures for finding the value – expressions of the other DBL constants .

Therefore , the DBL expressions are actually evaluable .
For the World Model Language and the English–oriented Formal Language , such a data base does not exist , but one could be imagined . We express this by saying that the WML and EFL expressions are " evaluable with respect to a virtual data base " .

4. Construction of the semantic deep structure of a question .

As we have seen , the English–oriented Formal Language differs from the other two languages in two respects :

1. It has different constants , of which the most important are :
   a. names of sets corresponding to nouns (e.g. " computers ") , to verbs (" buy – situations ") and to some of the prepositions (" in – place – situations ") .
   b. grammatical functions : subject , object , etc .
2. It has some different constructions . Here the most striking difference is that EFL constructions contain semantic and syntactic features . The semantic features influence the formal semantics of the constructions (e.g. the definiteness or indefiniteness of a noun phrase influences the choice of the kind of quantification for that noun phrase) . The syntactic features only play a role during the transformation process from English to EFL .

It should be noted that in general two synonymous sentences need not be represented by the same semantic deep structure in EFL . For example , the synonymy of ' A buys B from C ' and ' C sells B to A ' is not accounted for at this level . However , at the level of the World Model Language synonymous sentences are mapped onto equivalent (not necessarily identical) WML expressions .
The construction of the semantic deep structure in EFL consists of three main phases :

phase 1: a lexicon , providing for each word one or more interpretations , represented by pairs (CAT_i , SEM_i) , where CAT_i is a syntactic category and SEM_i an EFL expression .

phase 2: a set of rules that enables to combine the sequence of pairs (CAT_i , SEM_i) , corresponding to the original sequence of words , into higher level categories and more complex structures , until we have ultimately the pair (SENTENCE , SEM_s) , where SEM_s is the EFL expression for the complete sentence .
A rule of phase 2 is a combination of a context free rule and a set of rules on EFL expressions , that show when and how a sequence of pairs

$$(CAT_1 , SEM_1) , \ldots\ldots , (CAT_k , SEM_k)$$

can be reduced to a pair (CAT_R , SEM_R) .
The general format of these rules is :

– context free reduction rule :

$$CAT_1 + \ldots\ldots\ldots + CAT_k \rightarrow CAT_R$$

– EFL rules :

$$\begin{bmatrix} COND_1 : \ldots\ldots\ldots \\ ACTION_1 : \ldots\ldots\ldots \end{bmatrix}$$

. . . . . .

. . . . . .

$$\begin{bmatrix} COND_n : \ldots\ldots\ldots \\ ACTION_n : \ldots\ldots\ldots \end{bmatrix}$$

The CONDi's are conditions on the EFL expressions SEM_1 , ... , SEM_k .
The $ACTION_i$'s show how a new EFL expression SEM_R can be constructed with the help of SEM_1 , ... , SEM_k . The rule is applicable if at least one of the conditions $COND_i$ is true . Then SEM_R is constructed according to $ACTION_i$ and the sequence of pairs is reduced to (CAT_R , SEM_R) . If more than one of the $COND_i$ is true , we have a local ambiguity .

phase 3: transformation rules that transform the semantic surface structure into an EFL expression that is called the semantic deep structure . These transformation rules handle aspects of meaning that could not be resolved locally , during phase 2. This applies for instance to anaphoric references and elliptic clauses in comparative constructions .

A simpler example is the specification of the subject in a clause like ' to use a computer ' . The semantic surface structure of this clause means: ' there is a use-situation , with some computer as its object , and an unspecified subject ' . Phase 2 can be said to ' disambiguate ' this expression in a context like ' when did Shell start to use a computer ? ' .

A transformation specifies the subject of the use-situation as ' Shell ' . This transformation would not apply if we had the verb ' propose ' instead of ' start ' .

The conditions of phase 2 and phase 3 contain a "shortcut" to the world model: the semantic types of the world model interpretations of the EFL constants are inspected in order to avoid the construction of semantic deep structures that have no interpretation in the world model . This blocks many unfruitful parsing paths .

5 . Translation from semantic deep structure to unambiguous World Model Language expression

The translation from a semantic deep structure (EFL expression) into an unambiguous World Model Language expression proceeds in 3 phases:

phase 1: Translation from EFL expression into ambiguous WML expression. In this phase , transformations are applied which replace expressions containing EFL constants by expressions containing WML constants . Their most conspicuous effect is the elimination of "situations" and "grammatical functions" . It is

important to note that the resulting expression often contains several "ambiguous constants". These arise from polysemous terms in English : words that have a "range" of possible meanings . Such terms lead now to expressions with ambiguous constants: constants that stand for a whole class of possible "instances". An expression containing such constants , stands for the class of well-formed expressions that can be generated by "instantiating" the ambiguous constants .

phase 2: Disambiguation of quantifications .

Many sentences are ambiguous with respect to quantification .

E .g . "Were the largest 3 computers bought by 2 French companies ?" can either ask whether there are 2 French companies such that they both bought each of these computers , or , perhaps more plausibly , it can ask whether there are 2 French companies such that together they bought these computers .

Until this stage in the process , the representation of such questions contains constructions which stand for both interpretations at once . But now that the system' s assumptions about the structure of the world are reflected in the expression , some such interpretations may be ruled out as implausible , because they would lead to the same answer , independent of what the state of affairs in the world is . E .g ., the first interpretation of the above example question has the value "false" , independently of the values of the constants in the expression . (Because the assumption that a computer can only be bought by one company was introduced by a previous transformation) . Therefore , the second interpretation is chosen .

phase 3: Disambiguation of WML constants .

The ambiguous WML constants can be instantiated in a very efficient manner by using the semantic type system: The possible interpretations of an ambiguous constant are severely restricted by the semantic types of the other constants that appear in its context.

6. Translation from World Model Language expression to Data Base Language expression

In the World Model Language , constants correspond to the concepts of the universe of discourse . In the Data Base Language , constants correspond to primitive logical and arithmetical procedures and to primitives of the data base . The choice of these primitives was governed by considerations of efficiency , rather than by the wish to represent neatly the structure of the universe of discourse . Therefore , WML and DB contain different constants .

The translation from a WML expression to the DBL expression that will be evaluated , proceeds in three stages :

1. Paraphrase of the WML expression , in order to eliminate " infinite notions ".
   WML contains constants representing infinite sets or infinite continua , like " integers " , " money-amounts " and " time ". Such constants can not be directly or indirectly represented in the data base , and hence have no DBL-translation . By paraphrasing the expression , the infinite notions can often be eliminated .
2. Translation of expressions containing WML constants into expressions containing DBL constants .
   This translation is required by phenomena like the following :
   - it is possible that a class of objects is not represented explicitly in the data base , while properties of its elements are represented indirectly , as properties of other , related objects . (E.g. , cities do not occur in the PHLIQA 1 data base , but their names are represented as the city-names of sites .)
     A special case of this phenomenon is the representation of a continuum by a class of discrete objects (E.g. , " core " is represented by " core memories ") :
   - objects may be represented more than once in the data base . E.g. , in the PHLIQA 1 data base , the file of computer users and the file of manufacturers

can contain records that represent one and the same firm .

– the data base is more limited than the world model . Some questions that can be expressed in WML can be answered only partially or not at all : the WML expression has no DBL translation . The present convertor detects such expressions and can generate a message which specifies what information is lacking .

Examples of this case are : the set " integers " (if the attempt of the previous convertor to eliminate it has been unsuccessful) , and the " date–of–taking–out–of–use " of a computer (which happens to be not in the data base).

3. Paraphrase of the DBL expression , in order to improve the efficiency of its evaluation .

The DBL expression produced by the previous convertor can already be evaluated , but it may be possible to paraphrase it in such a way , that the evaluation of the paraphrase expression is more efficient . This conversion is worthwhile because , even with our small data base , the evaluation is often the most time–consuming part of the whole process ; compared to this , the time that transformations take is negligible .

7. The evaluation of a Data Base Language expression

The value of a Data Base Language expression is completely defined by the semantic rules of the Data Base Language (see section 3 . 2 .) , and one could conceive of an algorithm that corresponds exactly to these rules . For reasons of efficiency , the actual algorithm differs from such an algorithm in some major respects :

– in evaluating quantifications over sets , it does not evaluate more elements of the set than is necessary for determining the value of the quantification .

– if (e.g. during the evaluation of a quantification) , a variable assumes a new value , this does not cause the re–evaluation of any subexpressions that don' t contain this variable .

Currently , evaluation occurs with respect to a small data base in core . To handle a real data base on disk , only the evaluation of constants would have to change .

8. PHLIQA 1 's Control Structure

The sections 4 through 7 sketched what the basic modules of the system (the " convertors ") do . We shall now make some very general remarks about the way they were implemented . These remarks apply to all convertors except the parser , which is described in some detail by Medema [1975] .

The convertors can be viewed as functions which map an input expression into a set of zero or more output expressions . Such a function is defined by a collection of transformations , acting on subexpressions of the input expression . Each transformation consists of a condition and an action . The action is applied to a subexpression if the condition holds for it . The action can either be a procedure transforming a subexpression to its " lower level equivalent " or it can be the decision " this subexpression cannot be translated to the next lower level " .

All convertors are implemented as procedures which operate on the tree that represents the whole question . The procedures cooperate in a " depth-first " manner : a conversion procedure finds successively all interpretations that the input expression has on the next lower level . For each of these interpretations , as soon as it is found , the next convertor is called. If no interpretation can be found , a message giving the reason for this " dead end " is buffered , and control is returned to the calling convertor .

If the answer is found , it is displayed . If requested , the system can continue its search for more interpretations . If the answer level is not reached , it displays the buffered message from the " lowest " convertor that was reached .

Colophon

The PHLIQA 1 program was written in SPL (a PL/I dialect) , and runs under the MDS time sharing system on the Philips P1400 computer of the Philips Research Laboratories at Eindhoven .
The quantification–disambiguation phase of the EFL–WML translation , the efficiency–conversion (step 3) in the WML–DBL translation , as well as some parts of the grammar , are not yet part of the running system , though the convertors are completely coded and the grammar is elaborately specified .
During the design of PHLIQA 1 , the PHLIQA project was coordinated by Piet Medema . He and Eric van Utteren designed the algorithmic structure of the system and made decisions about many general aspects of implementation .
The formal languages and related transformation rules were designed by Harry Bunt . Jan Landsbergen and Remko Scha . Wijnand Schoenmakers designed the evaluation component . Jan Landsbergen wrote a grammar for an extensive subset of English.
All authors were involved in the implementation of the system .
During the design of PHLIQA 1 , extensive discussions with members of the SRI Speech Understanding team have helped us in making our ideas more explicit .

References

CODASYL Data Base Task Group
April 71 report . ACM , New York , 1971 .

P. Medema
A control structure for a question answering system .
Proceedings of the 4th International Joint Conference on Artificial Intelligence . Tbilisi , USSR , 1975. Vol. 2 .

S. R. Petrick
Semantic Interpretation in the REQUEST system .
Proceedings of the International Conference on Computational Linguistics , Vol. 1 , Pisa , 1973 .

W. J. Plath
Transformational Grammar and Transformational Parsing in the REQUEST system .
Proceedings of the International Conference on Computational Linguistics , Vol. 2 , Pisa , 1973 .

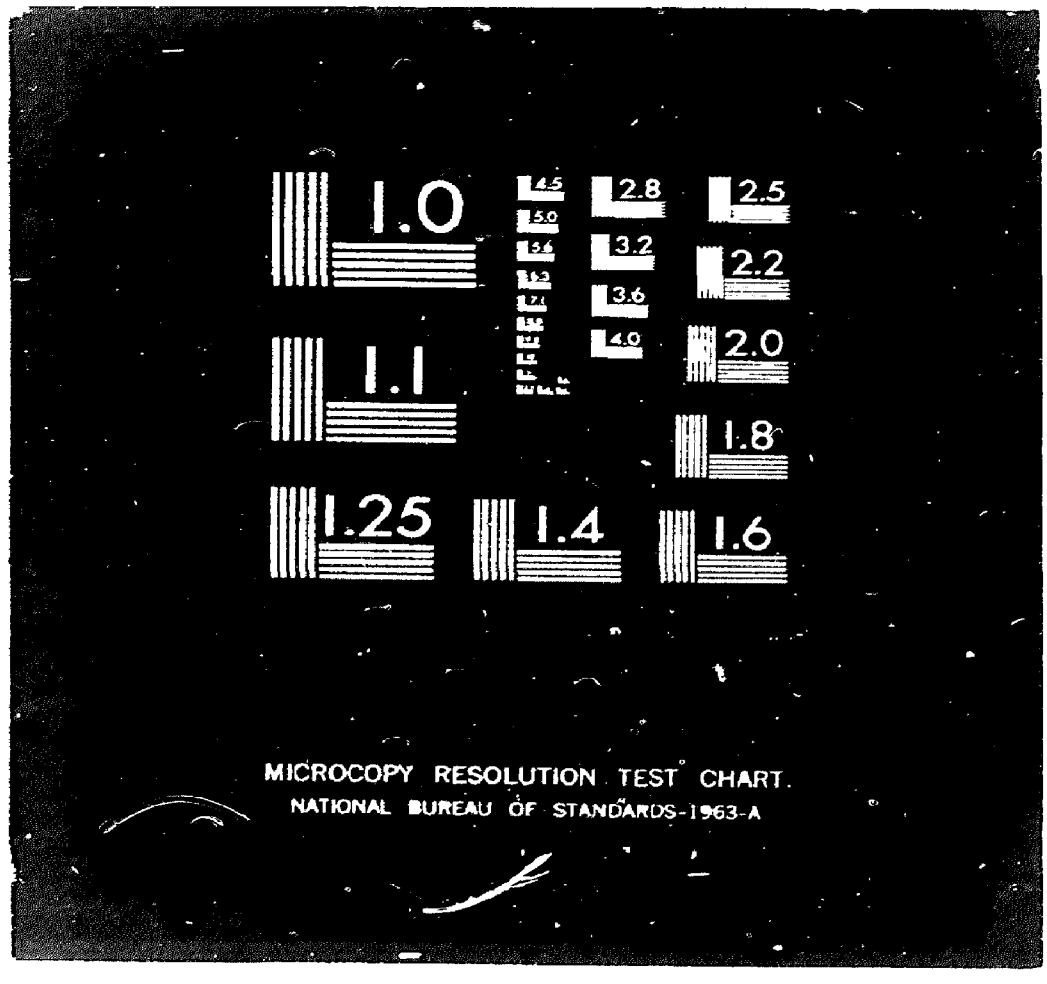
J. A. Robinson
Mechanizing Higher–Order Logic .
In : B. Meltzer and D. Michie (eds.) ,
Machine Intelligence 4 , Edinburgh University Press , 1969.

P. Wegner
The Vienna Definition Language .
Computing Surveys , Vol. 4 , no. 1 , 1972 .

T. Winograd
Understanding Natural Language .
Cognitive Psychology , Vol. 3 , no. 1 , 1972 .

W. A. Woods , R. M. Kaplan and B. Nash–Webber
The Lunar Sciences Natural Language Information System : Final Report . BBN , Cambridge , Mass. 1972 .

END


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
MICROCOPY RESOLUTION TEST CHART.
NATIONAL BUREAU OF STANDARDS-1963-A