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 assess 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 different 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 syntactic 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

\[ P \implies (\lambda x_1, \ldots, x_n: Q), \]

now the rule scheme

\[ P (x_1, \ldots, x_n) \implies Q \]

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 \(^1\):

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

The procedure \text{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: \( A[x_i := a_1, \ldots, x_n := a_n] \) stands for the expression \( A \) in which variable \( x_i \) has been
replaced throughout by expression \( a_1 \), \ldots, 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? \hspace{1cm} (1)

\textbf{Count}\left(\{x \in \text{DEPARTMENTS} \mid \text{Count}\left(\{y \in \text{EMPLOYEES} \mid \text{Have}(x, y)\}\right) > 100\}\right) \hspace{1cm} (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

\textbf{What is the total number of employees of all departments?} \hspace{1cm} (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.
\( \text{Count} \left( \{ x \in \text{DEPTS} \mid \#\text{EMP} \left( x \right) > 100 \} \right) \) \hspace{1cm} (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 \( \text{id}_{\text{emp}} \left( i \right) \), 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 \( \text{DIDS} \), which assigns to any department the identification numbers of its employees. Note that the extension of \( \text{DIDS} \) is unknown. For an expression to be evaluable, \( \text{DIDS} \) must be eliminated. Two properties of \( \text{DIDS} \) are important:

- For two distinct departments \( a \) and \( b \), \( \text{DIDS} \left( a \right) \) and \( \text{DIDS} \left( b \right) \) are not necessarily disjoint.
- For any department \( d \), \( \text{Count} \left( \text{DIDS} \left( d \right) \right) = \#\text{EMP} \left( d \right) \).

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

\[
\text{EMPLOYEES} \Rightarrow \bigcup \text{(for: DEPTS, apply: DIDS)}
\]

\[
\text{DEPARTMENTS} \Rightarrow \text{DEPTS}
\]

\[
\text{HAVE} \Rightarrow \left( \lambda u, v : v \in \text{DIDS} \left( u \right) \right)
\]

\[
\text{GS}_{\text{entity}} \Rightarrow \bigcup \text{(DEPTS, } \bigcup \text{(for: DEPTS, apply: DIDS)})
\]

Applying these rules to (2) yields:

\[
\text{Count} \left( \{ x \in \text{DEPTS} \mid \text{Count} \left( \{ y \in \text{(for: DEPTS, apply: DIDS)} \mid \ y \in \text{DIDS} \left( x \right) \} \right) > 100 \} \right)
\]

which is equivalent to:

\[
\text{Count} \left( \{ x \in \text{DEPTS} \mid \text{Count} \left( \{ y \in \text{DIDS} \left( x \right) \} \right) > 100 \} \right)
\]

which is equivalent to:

\[
\text{Count} \left( \{ x \in \text{DEPTS} \mid \text{Count} \left( \text{DIDS} \left( x \right) \right) > 100 \} \right)
\]

which is equivalent to:

\[
\text{Count} \left( \{ x \in \text{DEPTS} \mid \#\text{EMP} \left( x \right) > 100 \} \right)
\]
Note that the last simplification, where \textbf{Count (DIDS (x))} was transformed into \#\textbf{EMP (x)}, was not a logical equivalence transformation, but involved a descriptive global rule scheme:

\[
\textbf{Count (DIDS (x))} \implies \#\textbf{EMP (x)}
\]

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,

\[
\textbf{Count (DIDS (x))} \implies \#\textbf{EMP (x)},
\]

is rewritten, by abstracting over the free variables, as

\[(\lambda x: \textbf{Count (DIDS (x))}) \implies (\lambda x: \#\textbf{EMP (x)}).
\]

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 \(A \implies B\) 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 \(\otimes\), \textbf{Count} \(\otimes\textbf{DIDS}\) can not be translated into \#\textbf{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 PHIQA1 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 PHIQA1 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

\[
\begin{align*}
A(B(x)) & \Rightarrow Q \\
B(C(x)) & \Rightarrow R
\end{align*}
\]

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:

\[
\text{result: } = \text{eval (translate} (x)\text{)}
\]

A possibly more efficient variant would be:

\[
\text{result: } = \text{eval (simplify (translate} (x)\text{))},
\]

\(^3\) 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) = def
  if A has the form 'equal(x₁, x₂)' then
    if EVAL(x₁) = EVAL(x₂) then TRUE else FALSE.
  else if A has the form 'conj(x₁, x₂)' then
    if EVAL(x₁) = FALSE then FALSE
    else if EVAL(x₂) = FALSE then FALSE else TRUE
  else if A has the form 'disj(x₁, x₂)' then
    if EVAL(x₁) = TRUE then FALSE
    else if EVAL(x₂) = TRUE then TRUE else FALSE
  else if A has the form 'application(x₁, x₂)' then
    if x₁ is a constant then the value which the data base function x₁ yields for the argument EVAL(x₂)
    else if x₁ has the form (λy: D)
      then EVAL(D[y := x₂])
  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) = def
  if there is an axiom of the form A = E then TRANSLATE(E)
  else if A has the form 'bc(A₁, ..., Aₙ)' where 'bc' is some syntactic construction of the logical language,
    then MAKE-BC(TRANSLATE(A₁), ..., TRANSLATE(Aₙ))
  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

\[
\text{eval}^* (X),
\]

where \( \text{eval}^* \) is an algorithm which attempts to evaluate a source-language expression directly, only calling the translation module when that is necessary. For instance, an \( \text{eval}^* \) algorithm running parallel to the \( \text{eval} \) algorithm of the previous subsection would be defined as follows.

\[
\text{eval}^* (A) = \text{def}
\]

\textbf{if} \( A \) is a constant and there is an axiom of the form \( A = E \) \textbf{then} \( \text{eval}^* (E) \)

\textbf{else if} \( A \) has the form 'equal \((x_1, x_2)\)' \textbf{then}

\textbf{if} \( \text{eval}^* (x_1) = \text{eval}^* (x_2) \) \textbf{then} \( \text{true} \) \textbf{else} \( \text{false} \)

\textbf{else if} \( A \) has the form 'conj \((x_1, x_2)\)' \textbf{then if} \( \text{eval}^* (x_1) = \text{false} \)

\textbf{then} \( \text{false} \)

\textbf{else if} \( \text{eval}^* (x_2) = \text{false} \)

\textbf{then} \( \text{false} \) \textbf{else} \( \text{true} \)

\textbf{else if} \( A \) has the form 'disj \((x_1, x_2)\)' \textbf{then if} \( \text{eval}^* (x_1) = \text{true} \) \textbf{then} \( \text{true} \)

\textbf{else if} \( \text{eval}^* (x_2) = \text{true} \)

\textbf{then} \( \text{true} \) \textbf{else} \( \text{false} \)

\textbf{else if} \( A \) has the form 'application \((x_1, x_2)\)' \textbf{then}

\textbf{if} \( x_1 \) is a TL-constant \textbf{then} the value which the data base function \( x_1 \) yields for the argument \( \text{eval}^* (x_2) \)

\textbf{else if} \( x_1 \) has the form '\( \lambda y : D \)'

\textbf{then} \( \text{eval}^* (D [y := x_2]) \)

\textbf{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

\[
\text{Ancestor} \implies (\lambda x,y: \text{Parent}(x,y) \lor (\exists z: \text{Parent}(z,y) \& \text{Ancestor}(x,z))
\]

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.

\[
\text{Ancestor} \implies (\lambda x,y: (\exists z: (\text{Ancestor}(x,z) \& \text{Parent}(z,y)) \lor \text{Parent}(x,y))
\]

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.


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 successfully 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(α) 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 → ∃ x P(x) by generating a formula α such that the conjunction of → P(α) 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. 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 \textit{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

\[ \forall x: P(x) = \lnot Q(x) \]  

The yes/no question represented by the formula

\[ P(a) \]
would be answered "no" on the basis of this knowledge, since under the Closed World Assumption

\[ \rightarrow P(a) \] (3)

is a member of the "implicit axiom collection". The yes/no question represented by the formula

\[ Q(a) \] (4)

would be answered "no" as well, since

\[ \rightarrow Q(a) \] (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, \( \rightarrow P(a) \) 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.\footnote{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.}

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
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 PHILQA1 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.

\[ \text{IDB} \quad A \; \text{teaches all calculus courses.} \quad (1a) \]

\[ \forall z \in \text{Calculus} : \; \text{Teach}(A, z) \quad (1b) \]

\(^6\) 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 \hspace{0.5cm} (2a)

\[ \forall y \in \text{CS}: \text{Teach}(B,y) \] \hspace{0.5cm} (2b)

If teacher \( u \) teaches course \( v \) and student \( w \) is enrolled in \( v \), then \( u \) is a teacher of \( w \). \hspace{0.5cm} (3a)

\[ \forall u \in \text{Teacher}: \forall y \in \text{Course}: \forall w \in \text{Student}: \]

\[ \text{Enrolled}(w,v) \land \text{Teach}(u,v) \Rightarrow \text{Teacher-of}(w,u) \] \hspace{0.5cm} (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?* \hspace{0.5cm} (4a)

\[ \{x \in \text{Teacher} \mid \text{Teacher-of}(a,x)\} \] \hspace{0.5cm} (4b)
To be able to treat a query with a resolution algorithm it must be put in the form:

\[ [x, T, P] \] \hspace{1cm} (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

\[ [x, \text{Teacher}, \neg \text{Teacher-of}(a,x)] \] \hspace{1cm} (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

\[ [x, \text{Teacher}, \neg \text{Enrolled}(a,y) \lor \neg \text{Teach}(x,y)] \] \hspace{1cm} (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

\[ [x, \{A\}, \neg \text{Enrolled}(a,z_{\text{calc}})] \] \hspace{1cm} (8)

Then against axiom (2), yielding

\[ [x, \{B\}, \neg \text{Enrolled}(a,y_{\text{calc}})] \] \hspace{1cm} (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)\} \quad (10)
\{x \in \text{Teacher} \mid \text{Enrolled}(a,v) \& \text{Teach}(x,v)\} \quad (11)
\{x \in \{A\} \mid \exists \ z \in \text{Calculus}: \text{Enrolled}(a,z)\} \quad (12)
\{x \in \{B\} \mid \exists \ y \in \text{Calculus}: \text{Enrolled}(a,y)\} \quad (13)

When evaluated these yield, respectively: \(\emptyset\), \(\{C\}\), \(\{A\}\), and \(\{B\}\). The union of these is presented as an answer:

\(\{A, B, C\}\) \quad (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

\[\{x \mid P(x)\} = \{b\} \lor \{x \mid P(x)\} = \{d\},\] \quad (15)

the mention-all question with content

\(\{x \mid P(x)\}\) \quad (16)

would have the disjunctive answer

\(\{b\}\) or \(\{d\}\) \quad (17)

The mention-one question

\(<x \mid P(x)>\) \quad (18)
would have as its only answer:

\[ \text{b or d.} \quad (19) \]

The mention-all answer which can be derived from this mention-one answer is:

\[ \{ \text{b} \} \text{ or } \{ \text{d} \} \text{ or } \{ \text{b,d} \} \quad (20) \]

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

\[ \text{Who are a's teachers?} \] (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:

\[ \{ \text{A, B, C} \} \text{ or } \{ \text{A, B, C, D} \} \quad (21) \]

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 \(-\text{Teacher-of (a,D)}\) 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 database 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

$$\{ x \in \text{Teacher} \mid \text{Teacher-of}(a,x) \} = \{ A, B, C \} \text{ or } \{ x \in \text{Teacher} \mid \text{Teacher-of}(a,x) \} = \{ A, B, C, D \}$$

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":

\[
\forall u \in \text{Teacher}: \forall v \in \text{Course}: \forall w \in \text{Student}:
\begin{align*}
\text{Enrolled}(w,v) & \land \text{Teach}(u,v) \Rightarrow \text{Teacher-of}(w,u) \\
\end{align*}
\]

(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.

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

\[
\begin{align*}
B_1 & \supset R(x_1, \ldots, x_n) \\
\ldots & \\
B_k & \supset R((x_1, \ldots, x_n)
\end{align*}
\]

(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,

\[
R(a_1, \ldots, a_n) \text{ is true iff }\]

\[
B_1[x_1 := a_1, \ldots, x_n := a_n] \text{ or } \ldots \text{ or } B_k[x_1 := a_1, \ldots, x_n := a_n].
\]

Instead of (22), one could therefore write

\[
R(x_1, \ldots, x_n) \equiv B_1 \lor \ldots \lor B_k
\]

(23)

or

\[
R \equiv (\forall x_1, \ldots, x_n: B_1 \lor \ldots \lor B_k).
\]

(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 \(<\text{Student, Teacher}\> \rightarrow \text{truthvalue}\).

At the intermediate level (Level 2) we have at least the constants
Enrolled', with type \(<\text{Teacher, Course}\> \rightarrow \text{truthvalue}\), and Teach', with type
\(<\text{Teacher, Course}\> \rightarrow \text{truthvalue}\).

At the lowest level (Level 3) we have at least the constants Calculus and
CS, both with type S(Course), Enrolled with type
\(<\text{Student, Course}\> \rightarrow \text{truthvalue}\), and Teach with type \(<\text{Teacher, Course}\> \rightarrow 
\text{truthvalue}\).

The generic sets of the atomic types are constants at every level:

\[ GS_{\text{Teacher}}, GS_{\text{Student}}, GS_{\text{Course}}. \]

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:

\[ \text{Teacher-of} \implies (\lambda w, u: \exists v \in GS_{\text{Course}}. \text{Enrolled'}(w,v) & \text{Teach'}(u,v)) \]

The translation from Level 2 to Level 3 contains the rules:

\[ \text{Enrolled'} \implies \text{Enrolled} \]

\[ \text{Teach'} \implies (kp,q: (p = A \ & \ q \in \text{Calculus}) \lor (p = B \ & \ q \in \text{CS}) \]
\[ \lor \text{Teach}(p,q)) \]

Consider the example query

\[ \{ x \in GS_{\text{Teacher}} \mid \text{Teacher-of}(a,x) \} \]

Application of the Level 1-to-Level 2 translation yields, after \(\lambda\)-reduction:

\[ \{ x \in GS_{\text{Teacher}} \mid \exists v \in GS_{\text{Course}}. \text{Enrolled'}(a,v) & \text{Teach'}(x,v) \} \]

Application of the Level 2-to-Level 3 translation yields after \(\lambda\)-reduction:
\{ x \in \text{GS}_{\text{Teacher}} \mid \exists \, v \in \text{GS}_{\text{Course}}: \text{Enrolled} \, (a, v) \& \\
\quad ((x = A \& \, v \in \text{Calculus}) \lor \\
\quad (x = B \& \, v \in \text{CS} \lor \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.


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.