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

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

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

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

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

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

The present chapter deals with the knowledge representation method underlying the PHLIQA1 algorithm although the details of the algorithm itself are not dealt with. In illustrating the use of the knowledge representation method, the intermediate levels used in the system will be ignored; only two levels of representation will be assumed. An English-oriented Formal Language EFL' which corresponds closely to the English
formulation for questions is the highest of these levels. A Data Base Language DBL which corresponds closely to a relational data base is used at the lower level.

An English-oriented level of meaning representation.

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

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

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

"How many departments have more than 100 employees?" \hspace{1cm} (1)

into

\[
\text{Count} \left( \{ x \in \text{DEPARTMENTS} \mid \text{Count} \left( \{ y \in \text{EMPLOYEES} \mid \text{HAVE} (x,y) \} \} > 100 \} \right)
\]

where both x and y have the type entity. 2)

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

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

A data base specifies an interpretation of a logical language by specifying the values of its descriptive constants. What these constants are, follows directly from the structure of the data base. To illustrate this, we shall use a simple relational data base. (In Chapter III, in discussing PHLQA1, a CODASYSYL data base was used as an example.) Assume that a data base has a file with records which represent the employees of a firm, and that this file has an attribute indicating the department of each employee. A file is also present with records which represent the departments of the firm. This part of the data base may be viewed as specifying the extension of a constant EMPS of type S(entity) standing for the set of employees, \(^3\) of a constant DEPTS of type S(entity) standing for all departments and of a function F-EMP-DEPT of type (entity → entity) assigning to every employee a department.

In terms of such a data base structure, question (1) may be formulated as (3):

\[
\text{How many departments have more than 100 employees?} \quad (1)
\]

\[
\text{Count}(\{x ∈ \text{DEPTS} | \text{Count}(\{y ∈ \text{EMPS} | F-\text{EMP-DEPT}(y) = x\}) > 100\}) \quad (3)
\]

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

The connection between EFL' and DBL.

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

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

\(^3\) This analysis of a relational data base is somewhat simpler than the one proposed in Chapter III where a many-sorted type system was used. The many-sortedness will be reintroduced in section 3 of the present chapter.
starting point for the gradual development of more sophisticated methods later on.

2. Conceptual Information in the Form of Translation Rules.

2.1. Translation Rules.

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

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

As an example application of this method, consider the example data base of the previous section. The data base language considered has the descriptive constants EMPS, with type S(entity), DEPTS, with type S(entity), and F-EMP-DEPT, with type (entity --> entity).

The EFL-to-DBL translation is now defined by the rules

\[
\begin{align*}
\text{DEPARTMENTS} & \quad \rightarrow \quad \text{DEPTS} \\
\text{EMPLOYEES} & \quad \rightarrow \quad \text{EMPS} \\
\text{HAVE} & \quad \rightarrow \quad (\lambda u, v : 
\text{F-EMP-DEPT}(v) = u)
\end{align*}
\]

These rules can be directly applied to the formula (2) which represents question (1). Substitution of the right hand expressions for the left hand constants in (2) yields (4), which is equivalent to (3) above.
How many departments have more than 100 employees?

\[ \text{Count}(\{x \in \text{DEPARTMENTS} \mid \text{Count}(\{y \in \text{EMPLOYEES} \mid \text{HAVE}(x,y)\}) > 100\}) \]

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

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

\[ \text{HAVE} \rightarrow (\lambda u,v: \text{F-EMP-DEPT}(v) = u). \]

Instead, one would have had to say something like

\[ \text{HAVE}(u,v) \rightarrow \text{F-EMP-DEPT}(v) = u. \]

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

2.2. Type constraints.

The definition of the logical language used in the PHILQA1 system (see Appendix A) does not allow arbitrary combinations of constants, variables and formal operators as language expressions. Since many combinations would in fact be meaningless, the language definition explicitly constrains the possible combinations of elements to the meaningful ones. The type system which accomplishes this task may be used in a similar way to constrain the allowable translation rules.

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

\[ \text{TYPE}(\text{CT}(c)) \subseteq \text{TYPE}(c). \]

The relation \(\subseteq\) between two types \(\alpha\) and \(\beta\) which is defined in Appendix A, section 5, implies that for every interpretation of the language the domain of \(\alpha\) is a subset of the domain of \(\beta\).

\(^4\) Global rules are used, for instance, in TQA (Petrick, 1982) and EUFID (Burger, 1977). When global rules are a little more complicated than the example above, they raise a completeness problem. It is difficult to know if the rules cover all the cases that can arise. Local rules make completeness issues much more tractable. The rest of this chapter demonstrates that global rules can be avoided more consistently than one might think. In Chapter VI, section 2 I shall take up the issue of local vs. global translation rules again.
To prove that this indeed guarantees the legitimacy of any target language expression resulting from the translation of a legitimate source language expression, the following property which holds for all branching categories of the logical language is needed: if a branching category which constructs an expression of type $\beta$ out of expressions of type $\alpha_1, \ldots, \alpha_n$ is applied to expressions of type $\gamma_1, \ldots, \gamma_n$ such that $\gamma_1 \sqsubseteq \alpha_1, \ldots, \gamma_n \sqsubseteq \alpha_n$, then the resulting expression has a type $\delta$ such that $\delta \sqsubseteq \beta$.

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

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

$$\text{TYPE}(\text{CT}(c)) \sqsubseteq \text{TYPE}(c).$$

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

$$\text{TYPE}(\text{ET}(e)) \sqsubseteq \text{TYPE}(e).$$

3. Translation between Languages with different Type Systems.

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

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

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

Because of this discrepancy between the type systems at the highest and the lowest level of the system, it is worthwhile to define synonymy-translations between languages which not only differ in the descriptive constants they contain, but also in the descriptive atomic types they have.
The translation specification as introduced in the previous section can also be used for this case. A list which specifies for every source language constant a synonymous target language expression defines, given a target language interpretation, at most one source language interpretation as being compatible with it: the interpretation which assigns to every descriptive constant of the source language SL the denotation of the corresponding expression of the target language TL, and to every atomic type \( \alpha \) of SL the denotation of the TL expression which corresponds to the SL constant \( \text{GS}_\alpha \).\(^5\)

(To every formal constant or type, the SL interpretation assigns the same entity as the TL interpretation.)

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

Given an atomic-type-translation \( \text{AT} \), the corresponding type-translation \( \text{TT} \) is defined as the function which assigns to any type \( \tau \) of SL the type of TL which is obtained by substituting for every atomic type \( \alpha \) in \( \tau \) the type \( \text{AT}(\alpha) \).

\( H \) is recursively defined as follows:

\[
H(e) = \begin{cases} 
  \text{CT}(e) & \text{if } e \text{ is a constant} \\
  \text{VT}(e) & \text{if } e \text{ is a variable} \\
  \begin{align*}
  (\lambda y: & D) \\
  \text{then } & (\text{if: } y \in A, \text{then: } D') \\
  \text{where } y & \equiv \text{VT}(x) \\
  A & \equiv H(\text{GENSET(TYPE}(x))) \\
  D' & \equiv H(D)
\end{align*}
& \text{else if } e \text{ has the form } (\lambda x: D) \\
  \begin{cases} 
  b(sel_1: e_1, \ldots, sel_n: e_n) & \text{else if } e \text{ has the form } b(sel_1: e_1', \ldots, sel_n: e_n') \\
  \text{where } e_1' & \equiv H(e_1), \ldots, e_n' \equiv H(e_n).
\end{cases}
\end{cases}
\]

\(^5\) We assume that the source-language contains for every atomic type \( \alpha \) a constant \( \text{GS}_\alpha \) denoting the domain of \( \alpha \).
In the definition of $H$, two functions were used which have not been introduced before: $VT$ and $GENSET$.

$VT$ may be any function which assigns to any variable $v$ of $SL$ a distinct variable $u$ of $TL$, such that $\text{type}(u) = \text{TT}(\text{type}(v))$.

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

\[
\text{GENSET} \ (\alpha) = \begin{cases} 
\text{GS}_\alpha & \text{if } \alpha \text{ is an atomic type} \\
\text{power} \ (\text{GENSET} \ (\alpha')) & \text{if } \alpha \text{ has the form } S(\alpha') \\
\text{bags} \ (\text{GENSET} \ (\alpha')) & \text{if } \alpha \text{ has the form } B(\alpha') \\
\text{files} \ (\text{GENSET} \ (\alpha')) & \text{if } \alpha \text{ has the form } F(\alpha') \\
\text{lists} \ (\text{GENSET} \ (\alpha')) & \text{if } \alpha \text{ has the form } L(\alpha') \\
\text{cartesian-product} \ (\text{GENSET} \ (\alpha_1), \ldots, \text{GENSET} \ (\alpha_n)) & \text{if } \alpha \text{ has the form } \langle \alpha_1, \ldots, \alpha_n \rangle \\
\text{functions}_i \ (\text{domain: GENSET} \ (\alpha_i), \text{range: GENSET} \ (\alpha_i)) & \text{if } \alpha \text{ has the form } (\alpha_a \rightarrow \alpha_v) \\
\text{functions}_p \ (\text{domain: GENSET} \ (\alpha_a), \text{range: GENSET} \ (\alpha_v)) & \text{if } \alpha \text{ has the form } (\alpha_a \rightarrow \alpha_v) \\
\text{union} \ (\text{GENSET} \ (\alpha_1), \ldots, \text{GENSET} \ (\alpha_n)) & \text{if } \alpha \text{ has the form } (\alpha_1, \ldots, \alpha_n) \\
\text{bag-to-set} \ (\text{for: cartesian-product} \ (\text{union} \ (\text{GS}_{\text{integer}}, \text{GS}_{\text{real}}), \text{GENSET} \ (\beta)), \text{apply: } (\lambda x: (\text{num: } x \ [1], \text{unit: } x \ [2]))) & \text{if } \alpha \text{ has the form AMT}(\beta) \\
\text{bag-to-set} \ ((\text{for: GENSET} \ (\beta), \text{apply: } (\lambda y: \text{id}, (y)))) & \text{if } \alpha \text{ has the form ID}_i (\beta)
\end{cases}
\]

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

In the definition of $H$, the translation of expressions of the form $(\lambda x: D)$ may be refined by giving separate consideration to the case that

\[H(\text{GENSET} \ (\alpha)) \equiv \text{GENSET} \ (\text{TT}(\alpha)),\]

where $\alpha$ is the type of $x$.

In this case, the simpler expression $(\lambda y: D')$ may be used instead of the translation

\[\lambda y: (\text{if: } y \in A, \text{then: } D'),\]

because the condition $y \in A$ necessarily has the value $\text{true}$. 
To guarantee that, for any expression $E$, $H(E)$ is a legitimate expression of TL, we require that for any constant $c$

\[
\text{TYPE}(\text{CT}(c)) \supset \text{TT}(\text{TYPE}(c)).
\]

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

- For any atomic type $\alpha$ of SL holds that all elements of $\text{COMPONENTS}(\text{AT}(\alpha))$ are atomic types of TL.

- For any two different atomic types $\alpha$ and $\beta$ of SL holds:

\[
\text{COMPONENTS}(\text{AT}(\alpha)) \cap \text{COMPONENTS}(\text{AT}(\beta)) = \emptyset.
\]

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

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

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

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

\[\text{gs}_{\text{emp}} \quad \text{gs}_{\text{dept}} \quad \text{f-emp-dept}\]

Notice that this requirement on the translation rules can be effectively checked. This makes it possible to do a "syntactic correctness test" on every new set of translation rules that is entered into a system which is under development. Such a test mode makes it possible to detect mistakes in the formulation of translation rules during the development of the system, rather than during test runs of a finished version. This idea, which parallels the idea of compile-time type checks in high-level programming languages, has greatly diminished the amount of debugging effort involved in realizing a faultless implementation of the PHILIIQA1 program.

The function $\text{COMPONENTS}$ is defined in Appendix A.

Because of the type constraints on translation rules mentioned above, the rule for $\text{H AV E}$ as formulated here would not have been allowed in the PHILIIQA1 framework. Instead, it would have been formulated as:

\[
\text{H AV E} \implies (\lambda u, v: \quad u = (\text{fun: function-choice} (\text{f-emp-dept}, (\lambda p: \text{false})))
\]

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

In this way it is avoided that a function is translated into a function with a smaller range, which can lead to semantically anomalous expressions. The final result is the same, however, as in the formulation above.
DEPARTMENTS  $\Rightarrow$  \( GS_{\text{dept}} \)
EMPLOYEES  $\Rightarrow$  \( GS_{\text{emp}} \)
HAVE
$\Rightarrow$  \((\lambda u, v: \text{F-EMP-DEPT}(v) = u) \quad \text{where } u \text{ ranges over the domain of dept and } v \text{ ranges over the domain of emp} \)
\( GS_{\text{entity}} \)
$\Rightarrow$  \( \cup (GS_{\text{emp}}, GS_{\text{dept}}) \)

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

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

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

where \( x \) and \( y \) range over the domain of \( \text{entity} \).

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

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

where both \( x' \) and \( y' \) range over the domain of \( \text{dept} \cup \text{emp} \).

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

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

4. Identification Translations.

4.1. The Problem of Compound Attributes.

If a target language does not have for every constant of the source language an expression considered synonymous to it, the relation between both languages cannot be defined by a synonymy translation. But it is nevertheless possible that in such a case a weaker, but interesting and useful relation exists between the languages. It may be possible to define an identification translation from the source language into the target language, in
which a translation assigns to every constant of the source language an
expression of the target language which represents a concept that can "do
duty" for the concept that the constant represents, without necessarily being
the same concept. (If it is the same concept in every case, the identification
translation is in fact a synonymy translation.)

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

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

"How many departments have more than 100 employees?" \hspace{1cm} (1)

\begin{align*}
\text{\textbf{Count}} & \left( \{ x \in \text{DEPARTMENTS} \mid \\
& \quad \text{\textbf{Count}} \left( \{ y \in \text{EMPLOYEES} \mid \text{H ave} \ (x,y) \} \} > 100 \} \right) . \\
(\text{Both } x \text{ and } y \text{ range over type } \text{entity}.) \hspace{1cm} (2)
\end{align*}

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

\begin{align*}
\text{\textbf{Count}} & \left( \{ x \in \text{GS}_{dept} \mid \text{\#EMP} \ (x) > 100 \} \right) . \\
(3)
\end{align*}

In describing the relation between EFL' and DBL for this case, a new
difficulty arises. The DBL constants do not allow the construction of DBL
expressions whose denotations involve employees. So the EFL' constant
EMPLOYEES cannot be translated into an equivalent DBL expression – nor can
the relation HAVE, for lack of a suitable domain. This may seem to require
giving up local translation for certain cases: instead, one would have to use an
algorithm which looks out for sub-expressions of the form
\hspace{1cm} (\lambda y: \text{\textbf{Count}} \ (\{ x \in \text{EMPLOYEES} \mid \text{H ave} \ (x,y) \})), \text{ where } y \text{ is ranging over}
\hspace{1cm} \text{DEPARTMENTS, and then translates this whole expression into: \#EMP. This}
\hspace{1cm} \text{would not be attractive – it could only work if EFL' expressions would be}
\hspace{1cm} \text{first transformed so as to always contain this expression in exactly this form,}
\hspace{1cm} \text{or if a large number of corresponding transformations for differently}
\hspace{1cm} \text{structured expressions would be specified. Happily, the "identification}
\hspace{1cm} \text{translation" provides another solution.}

Although in DBL terms one cannot talk about employees, one can talk
about objects which stand in a one-to-one correspondence to the employees:
the pairs consisting of a department \( d \) and a positive integer \( i \) such that \( i \) is not larger than the value of \( \#\text{EMP} \) for \( d \) (see fig. 1.).

\[
\begin{array}{ccc}
\text{DEPT A} & & \text{DEPT B} \\
\#\text{EMP}: 3 & & \#\text{EMP}: 4 \\
\langle A,1\rangle & & \langle B,1\rangle \\
\langle A,2\rangle & & \langle B,2\rangle \\
\langle A,3\rangle & & \langle B,3\rangle \\
\langle A,4\rangle & & \langle B,4\rangle \\
\end{array}
\]

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

\begin{align*}
\text{EMPLOYEES} & \Rightarrow \cup (\text{for: } \text{GS}_{\text{dept}}, \text{apply: } (\lambda x: \{x\} \times \text{Int} (\#\text{EMP}(x)))) \\
\text{DEPARTMENTS} & \Rightarrow \text{GS}_{\text{dept}} \\
\text{HAVE} & \Rightarrow (\lambda u, v: u = v [1]) \\
\text{GS}_{\text{entity}} & \Rightarrow \cup (\text{GS}_{\text{dept}}, \cup (\text{for: } \text{GS}_{\text{dept}}, \\
& \quad \quad \text{apply: } (\lambda x: \{x\} \times \text{Int} (\#\text{EMP}(x))))))
\end{align*}

(\text{Int} is a function which assigns to any integer \( i \) the set of those integers \( j \) such that \( 0 < j \leq i \).)

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

\[
\text{Count } \{(z \in \text{GS}_{\text{dept}}) | \text{Count } \left( \{y \in \cup (\text{for: } \text{GS}_{\text{dept}}, \\
\text{apply: } (\lambda x: \{x\} \times \text{Int} (\#\text{EMP}(x))) | y[1] = z \} \right) > 100 \} \tag{4}
\]

This expression can in fact be shown to be equivalent to (3) above. In this case, this method yields a correct result. There are problems with the method nevertheless.
Notice that EFL' individuals are translated into compound DBL entities. This means that in EFL' it would be necessary to give up the idea that atomic types and compound types have disjoint domains. This may cause technical problems, in that the applicability conditions of some important simplification transformations have to be reconsidered. Another, more serious problem is the fact that the correctness of the method just illustrated depends on global properties of the formula that it is applied to, properties which have not been made explicit. This method therefore can not be guaranteed to work in every case. If the source language formula, for instance, would contain a condition to the effect that a certain employee would equal a certain pair \(<\textit{department}, \textit{integer}\>) (a condition which would be logically equivalent to FALSE), then this condition would be translated into one which for some data bases would come out TRUE.

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

\[
\text{EMPLOYEES} \quad \Rightarrow \quad (\text{for: } \cup (\text{for: } G_{S_{\text{dept}}}, \\
\quad \quad \text{apply: } (\lambda x: \{x\} \times \text{Ints} (\#\text{EMP} (x))), \\
\quad \quad \text{apply: } (\lambda y: \text{id}_{\text{emp}} (y))) \\
\text{DEPARTMENTS} \quad \Rightarrow \quad G_{S_{\text{dept}}} \\
\text{HAVE} \quad \Rightarrow \quad (\lambda u, v: u = \text{rid} (v) [1]) \\
\text{GS}_{\text{entity}} \quad \Rightarrow \quad \cup (G_{S_{\text{dept}}} (\text{for: } \cup (\text{for: } G_{S_{\text{dept}}}, \\
\quad \quad \text{apply: } (\lambda x: \{x\} \times \text{Ints} (\#\text{EMP} (x))), \\
\quad \quad \text{apply: } (\lambda y: \text{id}_{\text{emp}} (y))))
\]

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

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

\[
\text{Count} \left( \{ z \in G_{S_{\text{dept}} } \mid \right.
\text{Count} \left( \{ y \in (\text{for: } \text{DEPTS}, \\
\quad \text{apply: } (\lambda x: \{x\} \times \text{Ints} (\#\text{EMP} (x))), \\
\quad \quad \mid \text{rid} (y) [1] = z ) \right) \right) > 100 \} \)
\]

(5)
where $x$, $y$ and $z$ range over $\cup (\text{dept, ID}_{\text{emp}} \langle\text{dept, integer}\rangle)$. (This is demonstrated in detail in section 5).


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

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

$\text{How many departments have more than 100 employees?}$ \hspace{1cm} (1)

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

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

\[
\begin{align*}
\text{EMPLOYEES} & \Rightarrow \cup (\text{for: OFFS} \\
& \quad \text{apply:} \ (\lambda x: (\text{for: Inths}\ (\#\text{EMP}\ (x))), \\
& \quad \text{apply:} \ (\lambda y: \text{id}_{\text{emp}}\ (\langle x, y \rangle))))
\end{align*}
\]

\[
\begin{align*}
\text{DEPARTMENTS} & \Rightarrow \text{DEPTS} \\
\text{ENTITIES} & \Rightarrow \cup (\text{DEPTS, ....}) \\
\text{HAVE} & \Rightarrow (\lambda u, v: \text{F-OFF-DEPT}\ (\text{rid}\ (v)\ [1]) = u)
\end{align*}
\]

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

\[
\begin{align*}
\text{Count}\ \{x \in \text{DEPARTMENTS} | \text{Count}\ \{y \in \text{EMPLOYEES} | \text{HAVE}\ (x, y)\} \\
& \quad > 100\}\) \hspace{1cm} (2)
\end{align*}
\]

\[
\begin{align*}
\text{Count}\ \{z \in \text{DEPTS} | \text{Sum}\ (\text{for:} \ {x \in \text{OFFS} | \text{F-OFF-DEPT}\ (x) = z}, \\
& \quad \text{apply:} \ \#\text{EMP}) > 100\}\)
\end{align*}
\]

\(^9\) OFFS $\equiv$ GS\text{off} and DEPTS $\equiv$ GS\text{dept}.
Thus, this example yields a more complicated DBL expression, where the values of the \#EMP attribute for the different offices of one department are added up.

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

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

\[
\text{How many employees do the departments have together?}
\]

(4)

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

\[
\text{Count} (\cup (\text{for: DEPARTMENTS, apply: } (\lambda x: \{ y \in \text{EMPLOYEES} | \text{have } (x, y) \})))
\]

(5)

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

\[
\text{Sum (for: DEPTS, apply: \#EMP)}
\]

(6)

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

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

4.3. The Definition of Identification Translations.

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

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

To capture the relation between the meanings of the constants and types of
SL and TL, AT and CT must fulfill the following intuitive requirements:

1. In every state of affairs of the subject domain, for every atomic type \( \alpha \) there is a one-to-one function \( f_\alpha \) which maps the extension of the concept represented by \( \alpha \) onto a subset of the extension of the concept represented by \( AT(\alpha) \). (In a particularly important special case, \( f_\alpha \) is simply the identity function). The idea behind this is that at the TL level elements of the domain of \( AT(\alpha) \) are used as "proxies" for the elements of the domain of \( \alpha \).

2. Either constant \( C \) and expression \( CT(C) \) represent the same concept, or \( C \) has a descriptive type\(^{10}\) and \( C \) and \( CT(C) \) represent "corresponding" concepts: concepts such that replacing every individual \( X \) (belonging to the domain of atomic type \( \alpha \)) in the extension of \( C \) by \( f_\alpha (X) \) necessarily yields the extension of \( CT(C) \).

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

1. a. For any atomic type \( \varphi \) of SL:
   \[ \exists X \in \text{COMPONENTS} (AT(\varphi)): (X \text{ is an atomic type } \nu \text{ } X \text{ has the form } \text{ID}_i(\gamma)). \]
   b. For any two distinct atomic types \( \varphi \) and \( \psi \) of SL:
   \[ \text{COMPONENTS} (AT(\varphi)) \cap \text{COMPONENTS} (AT(\psi)) = \emptyset \text{ } \forall \]
   \[ \exists i \exists x \in \text{COMPONENTS} (AT(\varphi)) \exists y \in \text{COMPONENTS} (AT(\psi)) \exists \delta: \ x \equiv \text{ID}_i(\gamma) \& \ y \equiv \text{ID}_i(\delta). \]
   c. For any formal atomic type \( \varphi \) of SL: \( AT(\varphi) \equiv \varphi. \)

2. a. For any constant \( C \) of SL:
   \[ \text{TYPE} (CT(C)) \overset{\pi}{\longrightarrow} \text{TT} (\text{TYPE} (C)), \]
   where the type translation function \( \pi \) is the function which assigns to any type \( \tau \) of SL a type of TL by replacing in \( \tau \) every occurrence of any atomic type \( \alpha \) by \( AT(\alpha) \).
   b. For any formal constant \( F \) of SL, \( CT(F) \equiv F \).

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

An interpretation \( I \) of TL and an interpretation \( J \) of SL are compatible iff\(^{11}\)

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

\(^{10}\) A descriptive type is a type which contains one or more descriptive atomic types.  
\(^{11}\) We use the notation "DOM_\(I(\alpha)\)" for "the domain of type \( \alpha \) under interpretation \( I \)" and "D_\(I(e)\)" for "the denotation of expression \( e \) under interpretation \( I \)".
The definition of the conversion function \( H \) remains the same as in the previous section, except for a precaution about the use of the \( \text{id}_i \)-branchings:

\[
H(e) = \begin{cases} 
\text{CT}(e) & \text{if } e \text{ is a constant} \\
\text{VT}(e) & \text{if } e \text{ is a variable} \\
\left( \lambda y : (\text{if } y \in A, \text{then } D') \right), & \text{if } e \text{ has the form } (\lambda x : D) \\
\text{where } y \equiv \text{VT}(x), & \\
A \equiv H(\text{GENSET}(\text{TYPE}(x)), & \\
D' \equiv H(D) & \\
\left[ e \text{ has the form } b(\text{sel}_1 : e_1, \ldots, \text{sel}_n : e_n) \right] & \\
\text{if } \exists i : (b \equiv \text{id}_i \& & \\
\text{there is a type in the range of AT which has the form } \text{ID}_i(\alpha)) & \\
\text{then } b(\text{sel}_1 : e_1', \ldots, \text{sel}_n : e_n'), & \\
\text{where } e_1' \equiv H(e_1), \ldots, e_n' \equiv H(e_n). & 
\end{cases}
\]

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

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

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

\(^{12}\) This is the case if a type of the form \( \text{ID}_i(\alpha) \), for this particular \( i \), occurs in the range of \( \text{AT} \).
5. Simplification Transformations.

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

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

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

\[
\text{EMPLOYEES} \rightarrow (\text{for: } \cup (\text{for: DEPTS, apply: } (\lambda x: \{x\} \times \text{Ints} (\#\text{EMP} (x))), \text{apply: } (\lambda y: \text{id}_{\text{emp}} (y))))
\]

This rule is to be preferred to the alternative formulation

\[
\text{EMPLOYEES} \rightarrow (\text{for: } \{x \in \text{DEPTS} \times \text{INTEGERS } | 0 < x [2] \leq \#\text{EMP} (x[1])\}, \text{apply: } (\lambda y: \text{id}_{\text{emp}} (y)))
\]

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

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

\[
\text{How many departments have more than 100 employees?}
\]

(1)

This question may be rendered in EFL' as

\[
\text{Count} \left( \{x \in \text{DEPARTMENTS} | \text{Count} \left( \{y \in \text{EMPLOYEES} | \text{have} (x, y)\} \right) > 100 \right) \right)
\]

(2)

where \(x\) and \(y\) have type entity.
Applying the identification translation as specified in section 4.1 yields

\[ \text{Count} \left( \{ z \in \text{DEPTS} \mid \right. \\]
\[ \quad \text{Count} \left( \{ y \in \text{(for: DEPTS, apply: } (\lambda x: \{ x \} \times \text{Ints (} \#\text{EMP}(x))\)), \right. \]
\[ \quad \quad \quad \text{apply: } (\lambda w: \text{id}_{\text{emp}} (w)) \}
\[ \quad \left. \mid \text{rid} (y [1] = z) > 100 \} \right) \]

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

(3)

This expression can be considerably simplified. I shall now show this step by step.

Applying to (3) the rule

\[ \{ y \in (\text{for: } A, \text{apply: } B) \mid C \} \implies (\text{for: } \{ z \in A \mid (\text{fun: } (\lambda y:C), \text{arg: } B(z)) \}, \]
\[ \quad \text{apply: } B) \]

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

\[ \text{Count} \left( \{ z \in \text{DEPTS} \mid \right. \]
\[ \quad \text{Count} \left( \{ (\text{for: } \{ u \in \cup(\text{for: DEPTS, apply: } (\lambda x: \{ x \} \times \text{Ints (} \#\text{EMP}(x))\))}, \right. \]
\[ \quad \quad \quad \mid \text{rid} (\text{id}_{\text{emp}} (u)) [1] = z \}, \]
\[ \quad \quad \text{apply: } \text{id}_{\text{emp}} \}
\[ \left. \quad > 100 \} \right) \]

(4)

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

Applying to (4) the rules

\[ \text{rid} (\text{id}_{\alpha} (x)) \implies x \]

and

\[ \{ u \in \cup(\text{for: } A, \text{apply: } B \mid C(u) \} \]
\[ \implies \cup(\text{for: } A, \text{apply: } (\lambda x: \{ z \in B(x) \mid C(z) \})) \]
yields

\[
\text{Count} \left( \{ z \in \text{DEPTS} \mid \right. \\
\hspace{1cm} \text{Count} \left( (\text{for: } \cup (\text{for: DEPTS}, \right. \\
\hspace{2cm} \text{apply: } (\lambda x: \{ q \in \{ x \} \times \text{Ints} (\#\text{EMP}(x)) \mid \right. \\
\hspace{3cm} q[1] = z)), \\
\hspace{4cm} \text{apply: } \text{id}_{\text{emp}})) \\
\hspace{5cm} > 100) \right) \right) 
\]

(5)

Applying to (5) the rule

\[
\{ q \in A \times B \mid q[1] = C \} \Rightarrow (\{ C \cap A \} \times B)
\]
yields

\[
\text{Count} \left( \{ z \in \text{DEPTS} \mid \right. \\
\hspace{1cm} \text{Count} \left( (\text{for: } \cup (\text{for: DEPTS}, \right. \\
\hspace{2cm} \text{apply: } (\lambda x: (\{ z \} \cap \{ x \}) \times \text{Ints} (\#\text{EMP}(x))) \right), \\
\hspace{3cm} \text{apply: } \text{id}_{\text{emp}}) \\
\hspace{4cm} > 100) \right) \right)
\]

(6)

Applying to (6) the rule:

\[
(\text{for: } A, \text{apply: } (\lambda x: (\{ x \} \cap B) \times C)) \\
\Rightarrow (\text{for: } A \cap B, \text{apply: } (\lambda x: \{ x \} \times C))
\]
yields:

\[
\text{Count} \left( \{ z \in \text{DEPTS} \mid \right. \\
\hspace{1cm} \text{Count} \left( (\text{for: } \cup (\text{for: DEPTS} \cap \{ z \}, \right. \\
\hspace{2cm} \text{apply: } (\lambda x: \{ x \} \times \text{Ints} (\#\text{EMP}(x))) \right), \\
\hspace{3cm} \text{apply: } \text{id}_{\text{emp}}) \\
\hspace{4cm} > 100) \right) \right)
\]

(7)

Applying to (7) the rule

\[
A \cap \{ x \} \Rightarrow \{ x \}
\]

if \( x \) has type \( \alpha \) and \( A \) is GS\( \alpha \),
yields:

\[
\text{Count}\left(\{ z \in \text{DEPTS} \mid \text{Count}\left(\text{for: } \forall\{z\} \\
apply\left(\lambda x: \{x\} \times \text{Ints}\left(\#\text{EMP}(x)\right)\right), \\
apply: id_{\text{emp}}\right) > 100\}\right) \right)
\] (8)

Applying to (8) the rule

\(\text{for: } \{z\}, \text{apply: } F \implies \{F(z)\}\)
yields:

\[
\text{Count}\left(\{ z \in \text{DEPTS} \mid \text{Count}\left(\text{for: } \forall\{z\} \times \text{Ints}\left(\#\text{EMP}(z)\right)\left(\right), \\
apply: id_{\text{emp}}\right) > 100\}\right)
\] (9)

Applying to (9) the rule

\(\forall (\{ A \}) \implies A\)
yields:

\[
\text{Count}\left(\{ z \in \text{DEPTS} \mid \text{Count}\left(\text{for: } \{z\} \times \text{Ints}\left(\#\text{EMP}(z)\right), \\
apply: id_{\text{emp}}\right) > 100\}\right)
\] (10)

Applying to (10) the rule

\(\text{Count}\left(\text{for: } A, \text{apply: } id_{\text{a}}\right) \implies \text{Count}(A)\)
yields:

\[
\text{Count}\left(\{ z \in \text{DEPTS} \mid \text{Count}\left(\{z\} \times \text{Ints}\left(\#\text{EMP}(z)\right)\left(\right), \\
apply: id_{\text{emp}}\right) > 100\}\right)
\] (11)

Applying to (11) the rule

\(\text{Count}\left(\{ z \} \times A\right) \implies \text{Count}(A)\)
yields

\[
\text{Count}\left(\{z \in \text{DEPTS} \mid \text{Count}\left(\text{Ints}\left(\#	ext{EMP}(z)\right) > 100\right)\}\right)
\]

Applying to (12) the rule

\[
\text{Count}\left(\text{Ints}(A)\right) \Rightarrow A
\]
yields

\[
\text{Count}\left(\{z \in \text{DEPTS} \mid \#	ext{EMP}(z) > 100\}\right)
\]

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

The last version of the PHLIQIA1 simplification module, designed and implemented by W.J.H.J. Bronnenberg, contained about 100 simplification rules which can be compared in complexity to the rules just demonstrated. The algorithm which attempts to simplify a PHLIQIA1 expression by applying these rules is rather complex. Two important problems giving raise to these complexities should be mentioned here.

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

6. Extending the Data Base Language.

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

\[
\text{How many departments have more than 100 employees?}
\]

\[
\text{Count}\left(\{x \in \text{DEPARTMENTS} \mid \text{Count}\left(\{y \in \text{EMPLOYEES} \mid \text{Have}(x,y)\}\right) > 100\}\right)
\]

Consider now a slight variation on (1):

\[
\text{How many departments have more than 100 people?}
\]
which is represented in EFL as

\[
\text{Count} \left( \{ x \in \text{DEPARTMENTS} \mid \text{Count} \left( \{ y \in \text{PEOPLE} \mid \text{HAVE} \ (x, y) \} \right) > 100 \} \right)
\]

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

\[
\text{Count} \left( \{ x \in \text{GS}_{\text{dept}} \mid \text{Count} \left( \{ y \in \text{GS}_{\text{emp}} \mid \text{F-EMP-DEPT}(y) = x \} \right) > 100 \} \right)
\]

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

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

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

\begin{align*}
\text{DEPARTMENTS} & \Rightarrow (\text{for: GS}_{\text{emp}}, \text{apply: F-EMP-DEPT}) \\
\text{EMPLOYEES} & \Rightarrow \text{GS}_{\text{emp}} \\
\text{PEOPLE} & \Rightarrow \cup (\text{GS}_{\text{emp}}, \text{GS}_{\text{nonemp}}) \\
\text{HAVE} & \Rightarrow (\lambda y, x: (\text{if: } x \in \text{GS}_{\text{emp}}, \text{then: F-EMP-DEPT}(x) = y, \text{else: FALSE})) \\
& \quad \text{where } y \text{ ranges over the domain of dept, and } x \text{ ranges over the domain of } \cup (\text{emp, nonemp}) \\
\text{GS}_{\text{entity}} & \Rightarrow \cup (\text{GS}_{\text{dept}}, \text{GS}_{\text{emp}}, \text{GS}_{\text{nonemp}})
\end{align*}
Application of these rules to (4) yields an expression which can be seen to be equivalent to (5).

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

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