Deriving Database Queries from Logical Forms by Abductive Definition Expansion

Manny Rayner and Hiyan Alshawi *
SRI International
Cambridge Computer Science Research Centre
23 Millers Yard, Cambridge CB2 1RQ, U.K.
manny@cam.sri.com hiyan@cam.sri.com

Abstract

The paper describes a principled approach to the problem of deriving database queries from logical forms produced by a general NL interface. Our method attempts to construct a database query and a set of plausible assumptions, such that the logical form is equivalent to the query given the assumptions. The domain information needed is provided as declarative meaning postulates, including "definitional equivalences". The technical basis for the approach is that a "definition" of the form $Head \wedge Conditions \leftrightarrow Body$ can be read procedurally as "Expand *Head* to *Body* if it occurs in an environment where *Conditions* can be inferred". The "environment" is provided by the other conjuncts occurring together with *Head* in the original logical form, together with other meaning postulates and the contents of the database. The method has been implemented in CLARE, a language and reasoning system whose linguistic component is the SRI Core Language Engine.

1 Introduction

The basic question addressed in this paper is that of how to connect a general NL interface and a back-end application in a principled way. We will assume here that the interface takes input in a natural language and produces a representation in some kind of enriched first-order logic, and that the application is some kind of relational database; this is a common and important situation, and it is well-known that the problems involved are non-trivial. The techniques used apply equally well to other NLP applications which involve mapping linguistic concepts to knowledge base predicates. Concrete examples in the paper will be taken from the SRI CLARE system, working in the domain of project resource management. CLARE is a combined natural language and reasoning system which includes the Core Language Engine (or CLE, Alshawi 1992) as its language component. The CLE produces semantic interpretations of sentences in a notation called Quasi Logical Form. For database interface applications, the semantic interpretations are converted into fairly conventional logical forms before query derivation takes place.

A NL interface like CLARE which is general (rather than being tailored to the application) will produce logical forms that essentially mirror the linguistic content of the input. It will thus normally contain what might be called "linguistic" predicates (i.e. word senses): for example, the logical form for a query like

(S1) List all payments made to BT during 1990.

would be expected to contain predicates corresponding directly to *payment*, *make* and *during*. An appropriate database query, on the other hand, might be a command to search for "transaction" tuples where the "payee" field was filled by "BT", and the "date" field by a date constrained to be between 1st January and 31st December, 1990. The differing nature of the two representations can lead to several possible kinds of difficulties, depending on how the "linguistic" and "database" representations are connected. There are three in particular that we will devote most of our attention to in what follows:

1. A query can be *conceptually* outside the database's domain. For example, if "payments" in (S1) is replaced by "phone-calls", the interface should be able to indicate to the user that it is unable to relate the query to the information contained in the database.
2. A query can be *contingently* outside the database's domain. Thus if "1990" is replaced by "1985", it may be possible to derive a query; however, if the database only contains records going back to 1989, the result will be an empty list. Presenting this to the user without explanation is seriously misleading.
3. A query may need additional implicit assumptions to be translatable into database form. Asking (S1) in the context of our example Project Resource Management domain, it is implicitly understood that all payments referred to have been made by SRI. If the user receives no feedback describing the assumptions that have been made to perform the translation, it is again possible for misunderstandings to arise.

*CLARE is being developed as part of a collaborative project involving BP Research, British Aerospace, British Telecom, Cambridge University, SRI International and the UK Defence Research Agency. The project is funded in part by the UK Department of Trade and Industry.

One attractive way to attempt to effect the connection between LF and database query is to encode the database as a set of unit clauses, and to build an interpreter for the logical forms, which encodes the relations between linguistic and database predicates as "rules" or "meaning postulates" written in Horn-clause form (cf. e.g. McCord 1987). Anyone who has experimented with this scheme will, however, know that it tends to suffer from all three of the types of problem listed above. This is hardly surprising, when one considers that Horn-clauses are "if" rules; they give conditions for the LF's being true, but (as pointed out in Konolige 1981), they lack the "only if" half that says when they are false. It is of course possible to invoke the Closed World Assumption (CWA); in this interpretation, finite failure is regarded as equivalent to negation. Unfortunately, experience also shows that it is extremely difficult to write meaning postulates for non-trivial domains that are valid under this strict interpretation.

For these reasons, Scha (1983) argues that approaches which express the connection between LF and database query in terms of first-order logic formulas are unpromising. Instead, previous approaches to query derivation which attempt to justify equivalence between queries and semantic represenations have been limited (at least in implemented systems) to employing restricted forms of inference. Examples are the type inference used in PHLIQA (Bronnenberg et al 1980) and Stallard's 'recursive terminological simplification' (Stallard 1986).

In this paper we will show how a more general deductive approach can be taken. This depends on coding the relationship between LF and database forms not as Horn-clauses but as "definitional equivalences", explicit if-and-only-if rules of a particular form. Our approach retains computational tractability by limiting the way in which the equivalences can take part in deductions, roughly speaking by only using them to perform directed expansions of definitions. However we still permit non-trivial goal-directed domain reasoning in justifying query derivation, allowing, for example, the translation of an LF conjuct to be influenced by any other LF conjuncts, in contrast to the basically local translation in PHLIQA. This approach deals with the first two points above without recourse to the CWA and simultaneously allows a clean integration of the "abductive" reasoning needed to take care of point 3. The main technical problems to be solved are caused by the fact that the left-hand sides of the equivalences are generally not atomic.

The rest of the paper is organized as follows. The main concepts are introduced in sections 2 and 3, followed by a simple example in section 4. Section 5 discusses the role of existential quantification in equivalences. In section 6 we introduce abductive reasoning, and relate this to the problems discussed above. Section 8 then briefly describes issues related to implementing efficient search strategies to support the various kinds of inference used, and in section 9 we present an extended example showing how an LF can be successively reduced by equivalences into DB query form.

2 Query Translation as Definition Expansion

The task which the CLARE database interface carries out is essentially that of translating a logical formula in which all predicates are taken from one set of symbols (word sense predicates) into a formula in which all predicates are taken from another set (database relations) and determining the assumptions under which the two formulae are equivalent. Since database relations are generally more specific than word senses, it will often be the case that the set of assumptions is non-empty. The same mechanism is used for translating both queries and assertions into database form; moreover, the declarative knowledge used is also compiled, using a different method, so as to permit generation of English from database assertions, though further description of this is beyond the scope of the paper.

The main body of the declarative knowledge used is coded in a set of equivalential meaning postulates in which word sense predicates appear on one side and database relations appear on the other. (In fact, intermediate predicates, on the way to translating from linguistic predicates to database predicates may appear on either side.) The translation process then corresponds to abductive reasoning that views the meaning postulates as conditional definitions of the linguistic predicates in terms of database (or intermediate) predicates, the conditions being either discharged or taken as assumptions for a particular derivation. We will therefore refer to the translation process as 'definition expansion'.

If the left-hand sides of equivalences needed to be *arbitrary* formulas, the whole scheme would probably be impractical. However, experimentation with CLARE has lead us to believe that this is not the case; sufficient expressive power is obtained by restricting them to be no more complex than existentially quantified conjunctions of atomic formulas. Thus we will assume that equivalential meaning postulates have the general form[1]

$$(\exists y_1, y_2, ...P_1 \wedge P_2 \wedge P_3...) \leftrightarrow P' \qquad (1)$$

In the implementation these rules are written in a notation illustrated by the following example,

```
exists([Event],
  and(work_on1(Event,Person,Project),
      project1(Project))) <->
DB_PROJECT_MEMBER(Project,Person)
```

in which `work_on1` and `project1` are linguistic predicates and `DB_PROJECT_MEMBER` is a database relation (we will adhere to the convention of capitalizing names of database relations).

The attractive aspect of this type of equivalence stems from the fact that it can be given a sensible interpretation in terms of the procedural notion of "definition-expansion". Neglecting for the moment the existential quantification, the intuitive idea is that is that (1) can be read as "P_1 can be expanded to P' if it occurs in

[1] Quantification over the y_i on the left-hand side will often in practice be vacuous. In this and other formulas, we assume implicit universal quantification over free variables.

an environment where $P_2 \wedge P_3 ...$ can be inferred". The "environment" is provided by the other conjuncts occurring together with P_1 in the original logical form, together with other meaning postulates and the contents of the database. This provides a framework in which arbitrary domain inference can play a direct role in justifying the validity of the translation of an LF into a particular database query.

3 Translation Schemas

The ideas sketched out above can be formalised as the inference rules (2), (3) and (4):

$$\begin{array}{ll} & (\exists y_1, y_2, ...P_1 \wedge P_2 \wedge P_3 ...) \leftrightarrow P' \wedge \\ & Conds \rightarrow \theta(P_2 \wedge P_3 ...) \\ \Rightarrow & Conds \rightarrow (\theta(P_1) \leftrightarrow P') \end{array} \quad (2)$$

where θ is a substitution that replaces each y_i with a different unique constant.

$$\begin{array}{ll} & Conds \wedge Q \rightarrow (P \leftrightarrow P') \\ \Rightarrow & Conds \rightarrow (P \wedge Q \leftrightarrow P' \wedge Q) \end{array} \quad (3)$$

$$\begin{array}{ll} & Conds \rightarrow (\theta(P) \leftrightarrow \theta(P'))) \\ \Rightarrow & Conds \rightarrow (\exists x.P \leftrightarrow \exists x.P')) \end{array} \quad (4)$$

where θ substitutes a unique constant for x.

In each of these, the formulas before the $\Rightarrow$ are the premises, and the formula after the conclusion. The inference rules can be justified within the framework of the sequent calculus (Robinson 1979), though space limitations prevent us from doing so here. (2) is the base case: it gives sufficient conditions for using (1) to expand P_1 (the head of the definition) to P' (its body). The other formulas, (3) and (4), are the main recursive cases. (3) expresses expansion of a conjunction in terms of expansion of one of its conjuncts, adding the other conjunct to the environment of assumptions as it does so; (4) expresses expansion of an existentially quantified form in terms of expansion of its body, replacing the bound variables with unique constants. We will refer to inference rules like (3) and (4) as *expansion-schemas* or just *schemas*. One or more such schema must be given for each of the logical operators of the representation language, defining the expansion of a construct built with that operator in terms of the expansion of one of its constituents.

The central use of the equivalences is thus as truth-preserving conditional rewriting rules, which licence translation of the head into the body in environments where the conditions hold. There is a second use of the equivalences as normal Horn-clauses, which as we soon shall see is also essential to the translation process. An equivalence of the form

$$P_1 \wedge P_2 \wedge \ldots \leftrightarrow Q_1 \wedge Q_2 \wedge \ldots$$

implies the validity, for any i, of all Horn-clauses either of the form

$$P_i \leftarrow Q_1 \wedge Q_2 \wedge \ldots$$

or

$$Q_i \leftarrow P_1 \wedge P_2 \wedge \ldots$$

We will refer to these, respectively, as *normal* and *backward* Horn-clause readings of the equivalence. For example, the rule

```
and(man1(X),employee1(X)) <->
exists([HasCar],employee(X,m,HasCar))
```

produces two normal Horn-clause readings,

```
man1(X) <- employee(X,m,HasCar).
```

```
employee1(X) <- employee(X,m,HasCar).
```

and one backward Horn-clause reading,

```
employee(X,m,sk1(X)) <- man1(X),employee1(X).
```

where `sk1` is a Skolem function. Note that in the equivalential reading, as well as in the backward one, it is essential to distinguish between existential and universal quantification of variables on the left-hand side. The equivalential reading of a rule of type

```
p(X,Y) <-> q(Y)
```

licences, for example, expansion of `p(a,b)` to `q(b)`; the justification for this is that `q(b)` implies `p(X,b)` for *any* value of `X`. However, if the rule is changed to

```
exists([X],p(X,Y)) <-> q(Y)
```

the expansion is no longer valid, since `q(b)` only implies that `p(X,b)` is valid for *some* value of `X`, and not necessarily for `a`. This pair of examples should clarify why the constants involved in schema (2) must be unique.

We are now in a position to explain the basic expansion process; in the interests of expositional clarity, we will postpone mention of the abductive proof mechanism until section 6. Our strategy is to use (2) and the expansion-schemas as the kernel of a system that allows expansion of logical forms, using the equivalences as expandable complex definitions.

The actual process of expansion of a complex formula F is a series of single expansion steps, each of which consists of the expansion of an atomic constituent of F. An expansion step contains the following sub-steps:

Recurse: descend through F using the expansion-schemas, until an atomic sub-formula A is reached. During this process, an environment E has been accumulated in which conditions will be proved, and some bound variables will have been replaced by unique constants.

Translate: find a rule $\exists y_i.(H \wedge C) \leftrightarrow B$ such that (i) H (the 'head') unifies with A with m.g.u. θ, and (ii) θ pairs the $\exists y_i$ *only* with unique constants in A deriving from existentially bound variables. If it is then possible to prove $\theta(C)$ in E, replace A with $\theta(B)$.

Simplify: if possible, apply simplifications to the resulting formula.

4 A Simple Example

We now present a simple example to illustrate how the process works.

In CLARE, the sentence (S2)

(S2) Do any women work on CLARE?

receives the LF

```
exists([C,E],
       and(woman1(C),work_on1(E,C,clare)))
```

This has to be mapped to a query which accesses two database relations, DB_EMPLOYEE(Empl,Sex,HasCar) and DB_PROJECT_MEMBER(Empl,Project); the desired result is thus:

```
exists([C,H],
       and(DB_EMPLOYEE(C,w,H),
           DB_PROJECT_MEMBER(clare,C)))
```

(Sex can be w or m). The most clearly non-trivial part is justifying the conversion between the linguistic relation woman1(X) and the database relation DB_EMPLOYEE(X,w,_). Even in the limited PRM domain, it is incorrect to state that "woman" is equivalent to "employee classed as being of female sex"; there are for example large numbers of women who are listed in the DB_PAYEE relation as having been the recipients of payments. It is more correct to say that a tuple of type DB_EMPLOYEE(X,w,_) is equivalent to the conjunction of two pieces of information: firstly that X is a woman, and secondly that she is an employee. This can be captured in the rule

```
and(woman1(Person),
    employee1(Person)) <->
exists([HasCar],
  and(DB_EMPLOYEE(Person,w,HasCar)))      (EQ1)
```

In the left-to-right direction, the rule can be read as "woman1(X) translates to DB_EMPLOYEE(X,w,_), in contexts where it is possible to prove employee1(X)." For the rule to be of use in the present example, we must therefore provide a justification for employee1(X)'s holding in the context of the query. The simplest way to ensure that this is so is to provide a Horn-clause meaning postulate,

```
employee1(X) <-
  DB_PROJECT_MEMBER(Project,X).           (HC1)
```

which encodes the fact that project members are employees.

Similarly, we will need an equivalence rule to convert between work_on1 and DB_PROJECT_MEMBER. Here the fact we want to state is that project-members are precisely people who work on projects, which we write as follows:

```
exists([Event],
  and(work_on1(Event,Person,Project),
      project1(Project))) <->
DB_PROJECT_MEMBER(Project,Person)         (EQ2)
```

We will also make indirect use of the rule that states that projects are objects that can be found in the first field of a DB_PROJECT tuple,

```
project1(Proj) <->
exists([ProjNum,Start,End],
  DB_PROJECT(Proj,ProjNum,Start,End))     (EQ3)
```

since this will allow us to infer (by looking in the database) that the predicate project1 holds of clare.

Two expansions now produce the desired transformation; in each, the schemas (4) and (3) are used in turn to reduce to the base case of expanding an atom. Remember that schema (4) replaces variables with unique constants; when displaying the results of such a transformation, we will consistently write X* to symbolize the new constant associated with the variable X.

The first atom to be expanded is woman1(C*), and the corresponding environment of assumptions is {work_on1(E*,C*,clare)}. woman1(C*) unifies with the head of the rule (EQ1), making its conditions employee1(C*). Using the Horn-clause meaning postulate (HC1), this can be reduced to DB_PROJECT_MEMBER(Project,C*). Note that C* in this formula is a constant, while Project is a variable. This new goal can now be reduced again, by applying the rule (EQ2) as a backwards Horn-clause, to

```
and(work_on1(Event,C*,Project),
    project1(Project))),
```

The first conjunct can be proved from the assumptions instantiating Project to clare; the second conjunct can now be derived from the normal Horn-clause reading of rule (EQ3), together with the fact that clare is listed as a project in the database. This completes the reasoning that justifies expanding woman1(C) in the context of this query, to

```
exists([HasCar],
  and(DB_EMPLOYEE(C,w,HasCar)))
```

The second expansion is similar; the atom to be expanded here is work_on1(E*,C*,clare), and the environment of assumptions is {woman1(C*)}. Now the rule (EQ2) can be used; its conditions after unification with the head are project1(clare), the validity of which follows from another application of (EQ3). So work_on1(E,C,clare) can be expanded to DB_PROJECT_MEMBER(clare,C), giving the desired result.

5 Existential Quantification

We have so far given little justification for the complications introduced by existential quantification on the left hand sides of equivalences. These become important in connection with the so-called "Doctor on Board" problem (Perrault and Grosz, 1988), which in our domain can be illustrated by a query like (S3),

(S3) Does Mary have a car?

This receives the LF

```
exists([C,E],
       and(car1(C),have1(E,mary,C))))
```

for which the intended database query will be

```
exists([S],
       DB_EMPLOYEE(mary,S,y))
```

if Mary is listed as an employee. However, we also demand that a query like (S4)

(S4) Which car does Mary have?

should be untranslatable, since there is clearly no way to extract the required information from the DB_EMPLOYEE relationship.

The key equivalence is (EQ4)

```
exists([E,C],
  and(car1(C),
      and(have1(E,P,C),
          employee1(P))) <->
exists([S],DB_EMPLOYEE(P,S,y))                (EQ4)
```

which defines the linguistic predicate **car1**. When used in the context of (S3), **(EQ4)** can be applied in exactly the same way as **(EQ2)** and **(EQ3)** were in the previous example; the condition **have1(E,P,C)** will be proved by looking at the other conjunct, and **employee1(mary)** by referring to the database. The substitution used to match the **car1** predication from the LF with the head of **(EQ4)** fulfills the conditions on the **translate** step of the expansion procedure: the argument of **car1** is bound by an existential quantifier both in the LF and in **(EQ4)**. In (S4), on the other hand, **car1** occurs in the LF in a context where its argument is bound by a **find** quantifier, which is regarded as a type of universal. The matching substitution will thus be illegal, and translation will fail as required.

6 Abductive Expansion

We now turn to the topic of abductive expansion. As pointed out in section 1, it is normally impossible to justify an equivalence between an LF and a database query without making use of a number of implicit assumptions, most commonly ones stemming from the hypothesis that the LF should be interpretable within the given domain. The approach we take here is closely related to that pioneered by Hobbs and his colleagues (Hobbs *et al* 88). We include declarations asserting that certain goals may be assumed without proof during the process of justifying conditions; each such declaration associates an *assumption cost* with a goal of this kind, and proofs with low assumption cost are preferred. So for example the meaning postulate relating the linguistic predicate **payment1** and the intermediate predicate **transaction** is

```
and(payment1(Trans),
    payment_from_SRI(Trans)) <->
exists([Cheque,Date,Payee],
  transaction(Trans,Cheque,Date,Payee))) (EQ5)
```

"transactions are payments from SRI"
and there is also a Horn-clause meaning postulate

```
payment_from_SRI(X) <-
   payments_referred_to_are_from_SRI.
```

and an assumption declaration

```
assume(payments_referred_to_are_from_SRI,
       cost(0))
```

The advantage of this mechanism (which may at first sight seem rather indirect) is that it makes it possible explicitly to keep track of when the assumption **payments_referred_to_are_from_SRI** has been used in the course of deriving a database query from the original LF. Applied systematically, it allows a set of assumptions to be collected in the course of performing the translation; if required, CLARE can then inform the user as to their nature. In the current version of the PRM application, there are about a dozen types of assumption that can be made. Most of these are similar to the one shown above: that is to say, they are low-cost assumptions that cheques, payments, projects and so on are SRI-related.

One type of assumption, however, is sufficiently different as to deserve explicit mention. These are related to the problem, mentioned in Section 1, of queries "contingently" outside the database's domain. The PRM database, for instance, is limited in time, only containing records of transactions carried out over a specified eighteen-month period. Reflecting this, meaning postulates distinguish between the two predicates **transaction** and **DB_TRANSACTION**, which respectively are intended to mean "A transaction of this type took place" and "A transaction of this type is recorded in the database". The meaning postulate linking them is

```
and(transaction(Id,CNum,Date,Payee),
    transaction_data_available(Date)) <->
DB_TRANSACTION(Id,CNum,Date,Payee)          (EQ6)
```

transaction_data_available is defined by the further postulate

```
transaction_data_available(Date) <-
   and(c_before(date(17,8,89),Date),
       c_before(Date,date(31,3,91)))        (HC2)
```

The interesting thing about **(HC2)** is that the information needed to prove the condition **transaction_data_available(Date)** is sometimes, though not always, present in the LF. It will be present in a query like (S1), which explicitly mentions a period; there are further axioms that allow the system to infer in these circumstances that the conditions are fulfilled. However, a query like (S5),

(S5) Show the largest payment to Cow's Milk.

contains no explicit mention of time. To deal with sentences like (S5), there is a meaning postulate

```
transaction_data_available(X) <-
   payments_referred_to_made_between(17/8/89,
                                     31/3/91).
```

with an associated assumption declaration

```
assume(
  payments_referred_to_made_between(17/8/89,
                                    31/3/91),
  cost(15)).
```

The effect of charging the substantial cost of 15 units for the assumption (the maximum permitted cost for an expansion step being 20) is in practice strongly to prefer proofs where it is not used; the net result from the user's perspective is that s/he is informed of the contingent temporal limitation of the database only when it is actually relevant to answering a query. This has obvious utility in terms of increasing the interface's user-friendliness.

7 Simplification Using Functional Information

A problem arising from the definition-expansion process which we have so far not mentioned is that the

database queries it produces tend to contain a considerable amount of redundancy. For example, we shall see below in section 9 that the database query derived from sentence (S1) originally contains three separate instances of the `transaction` relation, one from each of the original linguistic predicates `payment1`, `make2` and `during1`. Roughly speaking, `payment1(Ev)` expands to `transaction(Ev,_,_,_)`, `make2(Ev,Ag,P,To)` to `transaction(Ev,_,To,_)` and `during_Temporal(Ev, Date)` to `transaction(Ev,_,_,Date)`; the database query will conjoin all three of these together. It is clearly preferable, if possible, to merge them instead, yielding a composite predication `transaction(Ev,_,To,Date)`.

Our framework allows an elegant solution to this problem if a little extra declarative information is provided, specifically information concerning functional relationships in predicates. The key fact is that `transaction` is a function from its first argument (the transaction identifier) to the remaining ones (the cheque number, the payee and the date). The system allows this information to be entered as a "function" meaning postulate in the form

```
function(transaction(Id,ChequeNo,Payee,Date),
        [Id] -> [ChequeNo,Payee,Date])
```

This is treated as a concise notation for the meaning postulate

$$\begin{aligned} & transaction(i, c_1, p_1, d_1) \\ \rightarrow \quad & (transaction(i, c_2, p_2, d_2) \leftrightarrow \\ & c_1 = c_2 \wedge p_1 = p_2 \wedge d_1 = d_2) \end{aligned}$$

which is just a conditional form of the equivalential meaning postulates already described. It is thus possible to handle "merging" simplification of this kind, as well as definition expansion, with a uniform mechanism. In the current version of the system, the transformation process operates in a cycle, alternating expansions followed by simplifications using the same basic interpreter; simplification consists of functional "merging" followed by reduction of equalities where this is applicable.

The simplification process is even more important when processing assertions. Consider, for example, what would happen to the pair of sentences (S6) - (S7) without simplification:

(S6) Clara is an employee who has a car.

(S7) Clara is a woman.

(S6) translates into the database form

```
exists([A,B],
       DB_EMPLOYEE(clara,A,y))
```

(The second field in `DB_EMPLOYEE` indicates sex, and the third whether or not the employee has a company car). This can then be put into Horn-clause form as

```
DB_EMPLOYEE(clara,sk1,y)
```

and asserted into the Prolog database. Since Clara is now known to be an employee, (S7) will produce the unit clause

```
DB_EMPLOYEE(clara,w,sk2)
```

The two clauses produced would contain all the information entered, but they could not be entered into a relational database as they stand; a normal database has no interpretation for the Skolem constants `sk1` and `sk2`. However, it is possible to use function information to merge them into a single record. The trick is to arrange things so that the system can when necessary recover the existentially quantified form from the Skolemized one; all assertions which contain Skolem constants are kept together in a "local cache". Simplification of assertions then proceeds according to the following sequence of steps:

1. Retrieve all assertions from the local cache.
2. Construct a formula A, which is their logical conjunction.
3. Let A_0 be A, and let $\{sk_1 \ldots sk_n\}$ be the Skolem constants in A. For $i = 1 \ldots n$, let x_i be a new variable, and let A_i be the formula $\exists x_i . A_{i-1}[sk_i/x_i]$, i.e. the result of replacing sk_i with x_i and quantifying existentially over it.
4. Perform normal function merging on A_n, and call the result A'.
5. Convert A' into Horn-clause form, and replace the result in the local cache.

In the example above, this works as follows. After (S6) and (S7) have been processed, the local cache contains the clauses

```
DB_EMPLOYEE(clara,sk1,y)

DB_EMPLOYEE(clara,w,sk2)
```

$A = A_0$ is then the formula

```
and(DB_EMPLOYEE(clara,sk1,y)
    DB_EMPLOYEE(clara,w,sk2))
```

and A_2 is

```
exists([X1,X2]
  and(DB_EMPLOYEE(clara,X1,y)
      DB_EMPLOYEE(clara,w,X2))
```

Since `DB_EMPLOYEE` is declared functional on its first argument, the second conjunct is reduced to two equalities, giving the formula

```
exists([X1,X2]
  and(DB_EMPLOYEE(clara,X1,y)
      and(X1 = w,
          y = X2))
```

which finally simplifies to A',

```
DB_EMPLOYEE(clara,w,y)
```

a record without Skolem constants, which can be added to a normal relational database.

8 Search Strategies for Definition Expansion

This section describes the problems that must be solved at the implementation level if the definition-expansion scheme is to work with acceptable efficiency. The structure of the top loop in the definition-expansion process is

roughly that of a Prolog meta-interpreter, whose clauses correspond to the "expansion-schemas" described in section 2.

The main predicate in the expansion interpreter contains an argument used to pass the environment of assumptions, which corresponds to the *Conds* in the schemas above. The interpreter successively reduces the formula to be expanded to a sub-formula, possibly adding new hypotheses to the environment of assumptions. When an atomic formula is reached, the interpreter attempts to find an equivalence with a matching head (where "matching" includes the restrictions on quantification described at the end of section 2), and if it does so then attempts to prove the conditions. If a proof is found, the atom is replaced by the body of the selected equivalence.

The computationally expensive operation is that of proving the conditions; since inference uses the equivalences in both directions, it can easily become very inefficient. The development of search techniques for making this type of inference tractable required a significant effort, though their detailed description is beyond the scope of this paper. Very briefly, two main strategies are employed. Most importantly, the application of "backward" Horn clause readings of equivalences is restricted to cases similar to that illustrated in section 4, where there are dependencies between the expansion of two or more conjuncts. In addition to this, there are a number of heuristics for penalizing expenditure of effort on branches judged likely to lead to infinite recursion or redundant computation.

For the project resource management domain, which currently has 165 equivalence rules, the time taken for query derivation from LF is typically between 1 and 10 seconds under Quintus Prolog on a Sun Sparcstation 2.

9 A Full Example

In this section, we will present a more elaborate illustration of CLARE's current capabilities in this area, showing how the process of definition expansion works for the sentence (S1). This initially receives an LF which after some simplification has the form

```
find([PayEv],
  exists([Payer,MakeEv],
    and(payment1(PayEv),
     and(make2(MakeEv,Payer,PayEv,bt),
         during1(PayEv,
                 interval(date(1990,1,1)),
                          date(1990,12,31))))))
```

As already indicated, the resulting database query will have as its main predicate the relation `DB_TRANSACTION(Id,ChequeNo,Date,Payee)`. We will also need an evaluable binary predicate `c_before`, which takes two representations of calendar dates and succeeds if the first is temporally before the second. The final query will be expressed entirely in terms of these two predicates.

The first step is to apply meaning postulates which relate the linguistic predicates `payment1`, `make2` to the intermediate predicate `transaction`. Recall, as explained in section 6 above, that `transaction` is distinct from `DB_TRANSACTION`. The relevant postulates are

```
and(payment1(Id),
   payment_from_SRI(Id)) <->
exists([C,D,Payee],
  transaction(Id,C,D,Payee))                    (EQ7)
```

"A payment from SRI is something that enters into a `transaction` relation as its first argument".

```
and(make2(Event,Assumed_SRI,Payment,Payee),
    and(payment_from_SRI(Event),
        transaction1(Payment))) <->
exists([C,D],
  and(transaction(Event,C,D,Payee),
      Event = Payment))                         (EQ8)
```

"A payment is made by SRI to a payee if it and the payee enter into a `transaction` relation as first and fourth arguments."

Note that the atom `payment_from_SRI(PayEv)`, occurring in the first two rules, will have to be proved using an abductive assumption, as explained in section 6. After (EQ7) and (EQ8) have been applied and the equality introduced by (EQ8) removed, the form of the query is

```
find([PayEv],
  exists([A,B,C,D,E],
    and(transaction(PayEv,A,B,C),
      and(transaction(PayEv,D,E,bt)),
        during1(PayEv,
                interval(date(1990,1,1)),
                         date(1990,12,31))))))
```

under the abductive assumption `payments_referred_to_are_from_SRI`.

The next rules to be applied are those that expand `during1`. The intended semantics of `during1(E1,E2)` are "E1 and E2 are events, and the time associated with E1 is inside that associated with E2". The relevant equivalences are now

```
during1(E1,E2) <->
exists([T1,T2],
  and(associated_time(E1,T1),
      and(associated_time(E2,T2),
          c_during(T1,T2))))                    (EQ9)
```

"The `during1` relation holds between E1 and E2 if and only if the calendar event associated with E1 is inside that associated with E2."

```
and(associated_time(Id,Date),
    transaction1(Id)) <->
exists([C,Payee,Y,M,D],
  transaction(Id,C,Date,P))                     (EQ10)
```

"`Date` is the event associated with a transaction event if and only if they enter into the `transaction` relation as third and first arguments respectively."

Applying (EQ9) and (EQ10) in succession, the query is translated to

```
find([PayEv],
  exists([Date,A,B,C,D,E,F,G],
   and(transaction(PayEv,A,B,C),
    and(transaction(PayEv,D,E,bt)),
     and(transaction(PayEv,F,Date,G),
```

```
    and(c_during(Date,
                 interval(date(1990,1,1)),
                          date(1990,12,31))))))
```

The query is now simplified by exploiting the fact that `transaction` is functional on its first argument: it is possible to merge all three occurrences, as described in section 7, to produce the form

```
find([PayEv],
  exists([ChequeId,Date],
    and(transaction(PayEv,ChequeId,Date,bt),
        c_during(Date,
                 interval(date(1990,1,1)),
                          date(1990,12,31)))))
```

Equivalences for temporal predicates then expand the second conjunct, producing the form

```
find([PayEv],
  exists([ChequeId,Date],
   and(transaction(PayEv,ChequeId,Date,bt),
    and(c_before(date(1990,1,1),Date),
        c_before(Date,date(1990,12,31))))))
```

Finally, `(EQ6)` above is applied, to expand the intermediate predicate `transaction` into the database relation `DB_TRANSACTION`.

When the `transaction` predication is expanded, `PayEv` and `Date` are replaced by corresponding constants `PayEv*` and `Date*`, as explained in section 2; the environment of assumptions is the set

```
{c_before(date(1990,1,1),Date*),
 c_before(Date*,date(1990,12,31)}
```

The relevant clauses are now `(HC2)` and a meaning postulate that encodes the fact that `c_before` is transitive, namely

```
c_before(Date1,Date3) <-
   c_before(Date1,Date2),
   c_before(Date2,Date3)                    (HC3)
```

By chaining backwards through these to the assumptions, it is then possible to prove that `transaction_date_available(Date*)` holds, and expand to the final form

```
find([PayEv],
  exists([ChequeId,Date],
   and(DB_TRANSACTION(PayEv,ChequeId,Date,bt),
    and(c_before(date(1990,1,1),Date),
        c_before(Date,date(1990,12,31))))))
```

This can be evaluated directly against the database; moreover, the system has managed to prove, under the abductive assumption `payments_referred_to_are_from_SRI`, that it is equivalent to the original query.

10 Conclusions and Further Directions

We believe that the definition-expansion mechanism provides a powerful basic functionality that CLARE will be able to exploit in many ways, some of which we expect to begin investigating in the near future. Several interesting extensions of the framework presented here are possible, of which we mention two.

Firstly, it can be the case that an expansion can only be carried out if a certain set of assumptions A is made, but that it is also possible to deduce the *negation* of one of the assumptions in A from the original LF. (For example, the query may refer to a time-period that is explicitly outside the one covered by the database). In a situation of this kind it is likely that the user has a misconception concerning the contents of the database, and will appreciate being informed of the reason for the system's inability to answer.

It is also fairly straight-forward to use the method to answer "meta-level" questions about the database's knowledge (cf. Rayner and Janson, 1987). For example, *Does the database know how many transactions were made in July?* can be answered affirmatively (relative to assumptions) if the embedded question *How many transactions were made in July?* can be expanded to an equivalent database query. We expect to be able to report more fully on these ideas at a later date.

References

Alshawi, H., ed. 1992. *The Core Language Engine.* Cambridge, Massachusetts: The MIT Press.

Bronneberg, W.J.H.J., H.C. Bunt, S.P.J. Landsbergen, R.J.H. Scha, W.J. Schoenmakers and E.P.C. van Utteren. 1980. "The Question Answering System PHLIQA1". In L. Bolc (ed.), *Natural Language Question Answering Systems.* Macmillan.

Hobbs, J.R., M. Stickel, P. Martin and D. Edwards. 1988. "Interpretation as Abduction". Proceedings of the 26th Annual Meeting of the Association for Computational Linguistics, 95–103

Konolige, K. 1981. *The Database as Model: A Metatheoretic Approch*, SRI technical note 255.

McCord, M.C. 1987. "Natural Language Processing in Prolog". In A. Walker (ed.) *Knowledge Systems and Prolog.* Addison-Wesley, Reading, MA.

Perrault, C.R. and B.J. Grosz. 1988. "Natural Language Interfaces". In *Exploring Artificial Intelligence: Survey Talks from the National Conferences on Artificial Intelligence*, Morgan Kaufmann, San Mateo.

Rayner, M. and S. Janson. 1987. "Epistemic Reasoning, Logic Programming, and the Interpretation of Questions". *Proceedings of the 2nd International Workshop on Natural Language Understanding and Logic Programming*, North-Holland.

Robinson, J.A. 1979. *Logic: Form and Function.* Edinburgh University Press.

Scha, R.J.H. 1983. *Logical Foundations for Question Answering*, Ph.D. Thesis, University of Groningen, the Netherlands.

Stallard, D.G. 1986. *A Terminological Simplification Transformation for Natural Language Question-Answering Systems.* Proceedings of the 24th Annual Meeting of the Association for Computational Linguistics, 241–246.