SRI International

The Database as Model: A Metatheoretic Approach

Technical Note 255

September 1981

By: Kurt Konolige
Computer Scientist

Artificial Intelligence Center
Computer Science and Technology Division

The work reported here was supported by the Defense Advanced Research Projects Agency under Contract N00039-80-c-0645. The views and conclusions contained in this document are those of the author and should not be interpreted as representative of the official policies of the Defense Advanced Research Projects Agency of the U.S. Government.

ABSTRACT

This paper presents a method of formally representing the information that is available to a user of a relational database. The intended application area is deductive question-answering systems that must access an existing relational database. To respond intelligently to user inquiries, such systems must have a more complete representation of the domain of discourse than is generally available in the tuple sets of a relational database. Given this more expressive representation, the problem then arises of how to reconcile the information present in the database with the domain representation, so that database queries can be derived to answer the user's inquiries. Here we take the formal approach of describing a relational database as the model of a first-order language. Another first-order language, the metalanguage, is used both to represent the domain of discourse, and to describe the relationship of the database to the domain. The formal advantages of this approach are presented and contrasted with other work in the area.

Contents

1. Introduction and Overview

This paper presents a method of formally representing the information that exists in a relational database. Database representation technology has progressed independently of Artificial Intelligence (AI), but there is a need to reconcile it with AI representations. Many AI systems that address real-world domains must access existing databases; typically these have been natural-language question-answering systems, e.g., LUNAR [18] and D-LADDER [9]. The D-LADDER database, for example, is a preexisting relational database that is distributed across several sites of the Arpanet; access to the information is through a predefined database query language. D-LADDER can parse and "understand" English-language questions for which no information is available from the database because it maintains a representation of the domain of discourse that subsumes the information present in the database. However, to respond reasonably to a user's question, D-LADDER must have some description of the information present in the database, and must be able to decide whether there is a database query that will answer the original question.[1] Other researchers, including Reiter [15] and Chang [1], have advanced solutions to the problem posed above; but their solutions are inadequate for a several reasons. A common fault is that they compromise the expressive power with which they represent the domain of discourse so as to arrive at a satisfactory algorithm for deriving database queries. We critique these systems more fully at the end of this paper.

In this paper we develop a representation rich enough to describe both the domain of discourse and the information that an associated database contains about that domain. Although the intended application area is deductive question-answering systems, we will

[1]Note that we are not making a judgment here about the utility of database representation technology. It may be the case that all the information present in the database under consideration could be more easily and conveniently represented using standard AI techniques, e.g., semantic nets or Horn clauses. The situations we are interested in are those where the database is a predefined part of the domain, and we must build some representation for the information it contains. Given the prevalence of databases as an information-storage technology, it seems reasonable to expect that some applied AI systems will have to interact with them.

argue at the end of the paper that the basic framework developed is applicable to a wide range of AI problems characterized by interaction between a user and a complex computer program.

2. The Database and the User

A database is a data structure that represents or *models* some aspect of the world. The subset of the world modeled by the database is called the ***domain of discourse.*** To see how the correspondence between the domain of discourse and the database is set up, and some of the problems that can be encountered, consider a simple database that consists of two tables. The first lists teachers and the courses that they teach:

TEACH

Teacher	Course
Smith	CS222
⋮	⋮

while the second table lists the advisors of students:

ADVISE

Advisor	Student
Smith	Clive
⋮	⋮

Now the way in which these tables model the domain of discourse (in this case, the relationships between students, teachers, and courses at some university) seems fairly straightforward. If there is a row in the **TEACH** table with a teacher's name and a course name, then in the domain of discourse the teacher actually teaches that course, e.g., Smith teaches CS222. Similarly, each row of the **ADVISE** table indicates that some teacher is the advisor of some student.

2.1 Representational Inadequacy

Unfortunately, tables such as **TEACH** and **ADVISE** do not let us say very much about a domain of discourse for which we do not have complete information. Informally, we will say that the database is *descriptively inadequate*, since it cannot characterize domains for which only partial information is available. The descriptive inadequacy of database representations can be broken down into two categories, which we call *inherent* and *pragmatic* inadequacies.

Inherent inadequacies stem from the nature of the database as a model of the domain of discourse. Typical examples of this type are the inability to express disjunctive facts (e.g., "Either Smith or Jones teaches CS222") and existential statements (e.g., "Some teacher advises Clive, but it isn't known which one"). These inadequacies are well-known, and we do not concentrate on them here.

Less well-known are pragmatic inadequacies. These arise from the way in which a user of the database tries to make sense of the database and its relation to the world (hence the term "pragmatic"). When the database is set up, the intent is to model a certain part of the world. The user, however, may not know the exact boundaries of this intended domain of discourse, and so may be surprised when the database does not have information that he requests, or may misinterpret the straightforward database replies to his queries.

The following catalog of pragmatic inadequacies is not intended to be exhaustive, but gives an informal notion of some of the representational issues that must be dealt with in designing natural-language interfaces to databases.

Absence of a Relation. The database may not contain a relation the user expects to find. For example, given that the **TEACH** and **ADVISE** tables give information on students and courses, the user may be tempted to ask, "What students are taking CS222?" Such a query simply cannot be phrased in terms of the given database tables. The ability to deal with this type of incompleteness is particularly important in building systems that must interact with users who are unfamiliar with the contents of a database.

Conceptual Mismatch. The database's domain of discourse may overlap the user's view of the world in complicated ways. For example, suppose the **TEACH** table's second column was not a course number, but simply "T" if the teacher was teaching some course, and "F" if he was not. The user would not expect to ask a question like "Is Smith related to courses by a 'T'?" Rather, he would still ask, "Is Smith teaching some course," and he would expect that the appropriate database query (involving Ts and Fs) would be generated for him.

Only-If Incompleteness. This type of incompleteness arises because the database may not contain complete information about what *isn't* in a particular relation. Suppose that the **TEACH** relation contains all known teacher-course pairs, but that some teachers may be teaching courses without being listed in the table. Then a question like "Is Smith teaching CS242?" cannot be answered in the negative if no entry is found in the **TEACH** table, because the entry may be absent even though Smith is, in fact, teaching CS242. This is called "only-if" incompleteness because a relationship in the domain of discourse holds *if* an entry exists in the table, but not necessarily *only if* the entry is there.

Domain Incompleteness. This inadequacy is similar to only-if incompleteness, but refers to domains of individuals, rather than to relationships among individuals. If a user asks "Are all teachers teaching courses?", it may not be possible to give an answer based on the information in the database. There may be teachers at the university who are not mentioned in either the **TEACH** or **ADVISE** tables; if that is the case, the database simply does not contain enough information to answer questions relating to all teachers.

2.2 The Problem and a Solution

Because the database is often an inadequate representation of the domain of discourse, we might consider using some other representational technology than the database. For example, a first-order language can deal with the inherent inadequacies discussed above, and there has been recent work in modal languages that overcome the pragmatic inadequacies [10]. In certain situations, however, it may not be possible to do away with the database in favor of some other representational technology; for instance, the database may already exist, and it would be too expensive to convert it to another representation, or it may be too large for current first-order theorem provers to deal with, or it may have good computational properties. So the problem we are addressing is not how to *replace* the database with another representation, but rather how to keep the database as a representational tool, while at the same time overcoming its descriptive inadequacy.

The obvious solution is to embed the database in a more comprehensive representational scheme. The bulk of simple facts about the domain of discourse will then reside in the database, while the more powerful representation (of which it is a part) can handle the more complex situations where the database is inadequate. In addition, there must be some process for deciding when a query applied to the more comprehensive representation can be transformed to a query applied solely to the database.

The structure of the solution is given in Figure 1: the database represents the part of the world that is labeled DOD_1; the user's view of the world is DOD_2, and overlaps the intended domain of the database. The more powerful representation describes both the user's view of the world and the relation of the database to this view. In this way we can capture the differences between the user's intended domain of discourse and the part of the world actually represented by the database. We will also be able to specify when a user's query refers only to the part of the world represented by the database, and hence give a formal specification of valid transformations between the more powerful representation and the database.

To formalize this solution, we introduce two different languages: one for the database and one for the more powerful representation. The database language is a first-order language (called DBL) for which we have a computable model, namely, the database. It is important that the DBL have a computable model; whenever the transformation process finds a DBL sentence equivalent to one in the more powerful representation, we know that we can evaluate its truth-value by a set of finite operations on the database. If we

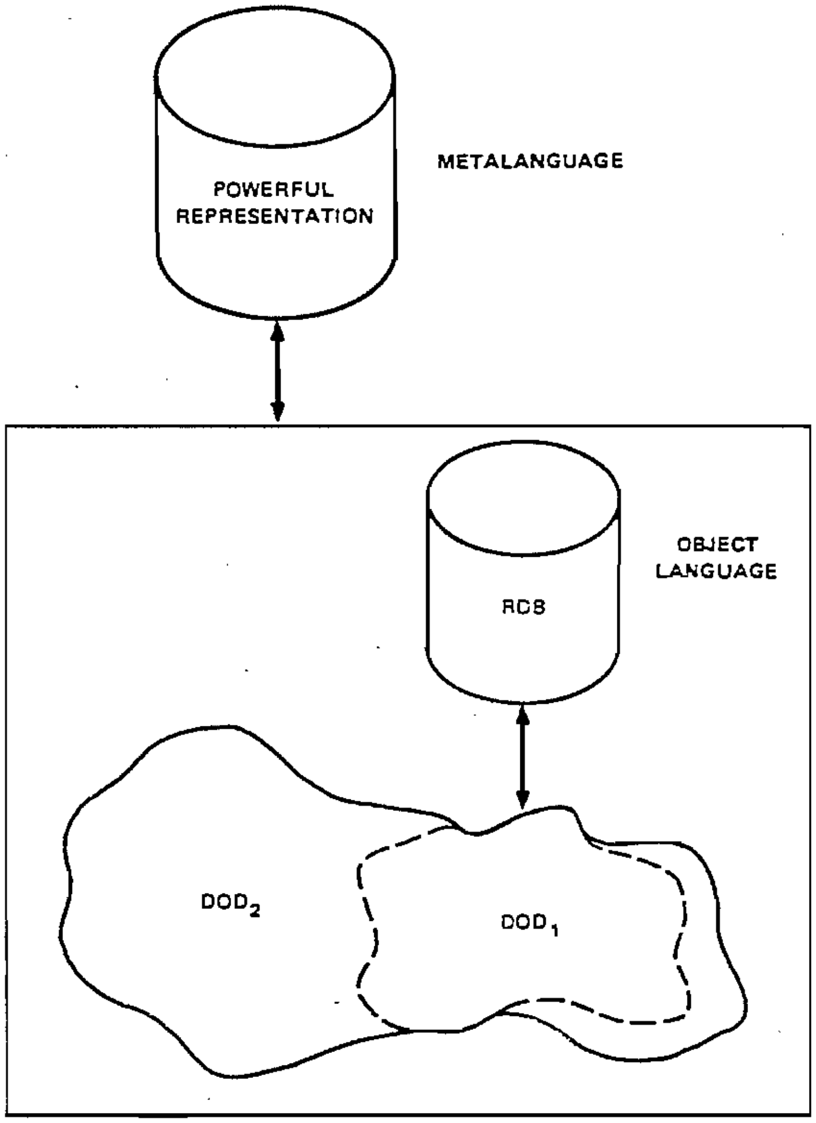

Figure 1. Metalanguage and Database Language

assume that we are using a relational database, the operations of the relational calculus can compute the truth-value for any sentence in the DBL. Queries to the database are expressed as sentences in the DBL; operations on the relations of the database can be used to find the truth-value of any formula of the DBL.

We will axiomatize the user's domain of discourse separately as a theory in a first-order predicate calculus with equality, which is a powerful descriptive formalism. This language is called the *metalanguage*, or ML (hereafter, we will use "domain of discourse" to refer to the intended domain of the ML rather than the DBL). In addition to encoding the domain of discourse, the ML theory will contain a description of the DBL. That is, the ML theory will have terms that denote DBL *expressions*, and a predicate that asserts the truth of DBL expressions in their intended model, the database. Finally, the ML theory will contain axioms that relate the truth of sentences in the DBL to predicates in the ML that describe the domain of discourse, thereby characterizing the connection between the database and the domain of discourse.

Queries about the domain are originally expressed as formulas in the ML. By attempting a constructive proof of a certain ML expression involving that formula, it is possible to either answer the query by a proof in the ML theory, or to find a DBL expression equivalent to the original ML query. In the latter case, because the DBL is fully interpreted by the database, the answer to the query can be found by evaluating the DBL expression with a suitable database query processor. This whole process is reminiscent of the *answer extraction* technique of Green [6].

In the next sections we formalize the framework just presented. First we define the database language DBL and show how a relational database is a model of this language. Then we define the metalanguage ML, and give examples of the connections between the ML and the DBL.

3. The Relational Database as a Model

For this paper, we restrict our attention to a particular type of database, the ***relational database***, because it is more readily amenable to formal analysis than are other database formulations.[2] A relational database is a set of relations $\{R_i\}$ over a set of domains $\{D_j\}$. Each relation R consists of a set of *tuples* of some fixed length n, the *arity* of the relation, which we indicate with a superscript, R^n. The elements of the tuples are drawn from the domains of the database.[3] In a relational database, the relations are finite, but the domains may be infinite, e.g., the domain *LENGTHS* in the sample database below is the set of positive real numbers over some interval.

To give an example of a relational database, consider the following relations about the ships world:

SHIPR:	$SHIPS \times SNAMES \times LENGTHS \times MEDIC$
COMMANDR:	$OFFICERS \times SHIPS$

The domains are *SHIPS*, *SNAMES* (ship names), *LENGTHS*, *OFFICERS*, and *MEDIC*. *MEDIC* is the binary domain $\{T, F\}$, and its use will be explained below. This sample database is derived from the *BLUE FILE* database accessed by the LADDER project [7]. A typical tuple of the SHIPR relation might be $\{USN123, LAFAYETTE, 344.6, T\}$.

A relational database is used to model the world in the following way. If a tuple $\{u, x, y, z\}$ is present in a relation R^4, then the objects u, x, y, and z are assumed to participate in the corresponding real-world relation. Thus the presence of the tuple $\{USN123, LAFAYETTE, 344.6, T\}$ in SHIPR means that the ship USN123 has the name Lafayette and the length 344.6. The interpretation of the *MEDIC* value is that the ship has a doctor on board if the value is T, and does not if the value is F.

[2]The approach described here will actually work with any database representation technology powerful enough to be a mechanizable model for some first-order language.

[3]This is unfortunate terminology because we have already introduced *domain of discourse*. The relational domains are sets of individuals drawn from the universe of individuals of the domain of discourse. Generally, there should be no confusion about the proper referent of *domain* in this paper because of context.

As discussed above, what happens if a tuple is not present in a relation depends on the interpretation one chooses for the database. The strongest assumption that can be made is that if a tuple is absent, the relation does not hold for the elements of that tuple. Whether this assumption is appropriate depends on the domain of discourse that is being modeled; it assumes that the database has complete information about the part of the world that corresponds to the relations it contains. In many applications, the database has only partial information about the domain of discourse. A formalization of the information that the database contains must be rich enough to represent a partial correspondence between the database's intended domain and the user's.

The relations in a relational database can be considered to form an algebraic structure under the operations join, restriction, projection, and set union and difference [3]. These operations can be used to extract information from the database conveniently. For example, suppose we want to find the officers of all ships over a length L. One way to do this would be to join tuples from the COMMANDR and SHIPR relations with the same *SHIPS* element,[4] restrict the resulting relation so that only tuples with a *LENGTHS* element greater than L remain, and then project the *OFFICERS* elements to yield a set of one-element tuples containing the answer.

A relational database can be formally described by designing an appropriate first-order language for it. The basic idea is that expressions in the language (which we will call the database language, or DBL) are either true or false with respect to the database; further, because we actually have the database in hand, the truth-value for any expression of the DBL can be determined by performing algebraic operations on the database. Thus, the DBL functions as a query language for the database because it describes properties of the database. A query is phrased as an expression in the DBL, and then algebraic manipulations can be performed on the database to determine the truth of the expression, and hence answer the query. Codd [3] has shown that the five given operations are sufficient to decide

[4]This operation is called an *equijoin*, and is the composition of a join with an equality restriction.

the truth-value of any expression in an appropriately defined DBL; hence, the database and its associated algebra form a computable or mechanizable model for the DBL.

At this point, it is helpful to look at the DBL for a particular database, the sample database given previously. We will use a many-sorted first-order language with equality, wih one sort for each relation in the database. A sort consists of all tuples in its corresponding relation; variables are restricted to range over a given sort. In quantified expressions, the sort a variable is restricted to will always be indicated by giving the name of the relation for the sort. For example, in

$$\forall t/_{\text{SHIPR}} \cdots$$

the variable t is restricted to tuples in the relation SHIPR. Because variables refer only to tuples in a relation, this type of language has been called a *tuple relational calculus* [16].

Besides variables over tuple sets, we allow function terms that refer to elements in the domain. Among these terms will be unary functions that pick out elements of a tuple. Generally, we use function names that are similar to the domain of the element they select from the tuple. As an example, consider:

$$\forall t/_{\text{SHIPR}}\ [sname(t) = LAFAYETTE] \vee [length(t) > 344.6].$$

The above expression says that for all tuples in the SHIPR relation, either the *SNAMES* element of the tuple is equal *LAFAYETTE*, or the *LENGTHS* element is greater than 344.6 meters. There are two unary functions, *sname* and *length*, and one constant function, *LAFAYETTE*.

The language also contains the boolean operators, the equality predicate, and arithmetic predicates such as *greater than* and *less than*. As defined, the DBL has the important property that every expression that can be written in the language is decidable with respect to the sample database, using the relational algebraic operations. Such a language is called *safe* in the database literature [16]; a safe language is one whose expressions can all be interpreted by examining only the instances of relations present in the database. The practical import of this is that safe languages are mechanizable, in the sense that the truth-value of every expression in the language can be determined by a finite number of algebraic manipulations on its intended model.

4. The Metalanguage

A language like the DBL for which the intended model is available is called an *interpreted language.* If the database correctly reflects the structure of the domain of discourse, we need no additional representational apparatus to describe the domain. However, it is more often the case that we have incomplete information about the domain of discourse: for example, we may know that the Lafayette is commanded either by Smith or Jones, without knowing which of the two is the actual commander. Such partial information about the domain cannot be expressed within the database model.

Another way to view this situation is to say that the database and the domain of discourse are both models of the DBL, but are not coextensive. That is, in the real world it may be the case that either Smith or Jones is the captain of the Lafayette; because we do not know which is the case, there will be no tuple in the COMMANDR relation of the form {*nnn*, *USN123*}, where *USN123* is the identifier of the Lafayette. So if the query "Does Jones command the Lafayette?" is posed in the DBL, the answer with respect to the database will be "no," which may not be the case in the actual world.[5]

If a question-answering system is to return correct responses to a user's queries when only partial information about the domain is available, then a more powerful representation than the DBL and its associated database is required. On the other hand, we want to use the information in the database in those cases where it is sufficient for responding to a query. So the representation we seek must not only characterize the domain of discourse, it must also encode the way in which the database as a model corresponds to the actual world.

[5]We assume in this paper that what information the database does contain about the domain of discourse is correct. In principle, the formalization presented in this section could readily handle cases where the database was not in conformity with the domain, e.g., the tuple {*JONES*, *USN123*} is present in the COMMANDR relation even when Jones is not actually the commander of the Lafayette. It may be useful in practice to have this ability, especially when dealing with a changing domain where updates to the database may not be timely.

To represent the domain of discourse, we use another many-sorted first-order language with equality, called the ***metalanguage***, or ML. The ML will have non-logical axioms that state properties about the domain. For example, for the ships world we might define the following predicates:

$DOC(x, y)$	means that x is the doctor aboard ship y
$NAME(x, y)$	means that x is the name of ship y
$LEN(x, y)$	means that x is the length of ship y
$COM(x, y)$	means that officer x commands the ship y.

A typical assertion about the domain might be:

$$\forall x/_{\text{SHIPS}}\, \exists y/_{\text{LENGTHS}}\, LEN(y, x),$$

which says that every ship has a length. That we have chosen this particular form of the first-order predicate calculus is not critical; any of the nonsorted variants would do just as well. Note that the ML need not be a tuple calculus; in this example, variables range over individual ships, officers, lengths, *etc.*, rather than over tuples.

In addition to representing the domain of discourse, the ML also characterizes the database as a model of the DBL; this is what makes it a *meta*language. In the ML, we use the predicate *DB* of one argument, a DBL formula, to mean that that formula holds in the database. Assume, for example, that the ML term f denotes the DBL formula $\forall t/_{\text{SHIPR}}\, [sname(t) = LAFAYETTE] \vee [length(t) > 344.6]$. Then the ML expression $DB(f)$ asserts that the DBL formula denoted by f is actually true of the database; that is, all tuples in the SHIPR relation either have their *sname* element equal to *LAFAYETTE*, or their *length* element greater than 344.6.

In the metalanguage, we require a number of constructors for DBL formulas. For boolean connectives of the DBL, the constructors *and*, *or*, *imp*, and *not* take DBL formulas as arguments and return the obvious DBL compound boolean formula. For example, the

ML term *and*(f, g) denotes the DBL formula $F \wedge G$, where F and G are the DBL formulas denoted by f and g. This ***abstract syntax*** for object-language formulas was introduced by McCarthy [12].

Because the DB predicate represents truth in the database model, the normal truth-recursion axioms are valid for it. We introduce the ML sort DBFS, which is the set of all DBL formulas, and write the truth-recursion axioms as:

$$\begin{aligned} \forall f/_{\text{DBFS}}\ &[DB(not(f)) \equiv \sim DB(f)] \\ \forall fg/_{\text{DBFS}}\ &[DB(or(f,g)) \equiv DB(f) \vee DB(g)] \\ \forall fg/_{\text{DBFS}}\ &[DB(and(f,g)) \equiv DB(f) \wedge DB(g)] \\ \forall fg/_{\text{DBFS}}\ &[DB(imp(f,g)) \equiv DB(f) \supset DB(g)] \end{aligned} \qquad TR1.$$

Additionally, to construct an arbitrary DBL formula we need ML terms that denote DBL predicates, terms, and quantifiers. At this point it is convenient to develop the theory for the propositional case; later it will be extended to predicates over individuals. For each DBL propositional constant there is a ML constant term that denotes it. The convention will be to use primed terms in the ML to denote the corresponding unprimed propositional constants in the DBL. Thus, the ML term P' will have the DBL proposition P as its denotation. Given this and the ML boolean constructors, it is possible to write a ML term for any sentential expression of the DBL, e.g., $and(P', not(Q'))$ denotes the DBL formula $P \wedge \sim Q$. The two ML terms $TRUE'$ and $FALSE'$ are specially defined to refer to the DBL propositions $TRUE$ and $FALSE$, whose truth-values in the database are always taken to be true and false, respectively:

$$\begin{aligned} DB(TRUE') &\equiv P \vee \sim P \\ DB(FALSE') &\equiv P \wedge \sim P \end{aligned} \qquad TR2.$$

5. Database Query Derivation

Having described the DBL and its associated database within the ML, we can formulate the derivation of database queries in the DBL from an original ML query. At this point we restrict our attention to ***closed queries***, that is, those whose answer is either yes or no. Closed queries can be represented by closed ML expressions.

Suppose that *qwff* is a ML expression whose truth-value is to be determined. Consider the ML schema:

$$\exists f/_{\mathrm{DBFS}}\ [DB(f) \equiv qwff] \qquad T1.$$

If a constructive proof of $T1$ can be found for a given instance of *qwff*, the binding for f will be a DBL formula equivalent to the original query *qwff*. By evaluating the DBL query with respect to the database, the truth of *qwff* can be determined. Note the use of the equivalence connective in $T1$: if the answer to the DBL language query turns out to be "false," then the negation of *qwff* will have been established.

A special case of $T1$ occurs if the binding for f is *TRUE* or *FALSE*. When this happens, the truth or falsity of *qwff* will have been established entirely within the metalanguage, without the necessity of evaluating a DBL query.

Example. Let P and Q be two DBL propositions whose meanings we intend to be the same as the ML predicates P and Q, respectively (we can use the same names because we always know which language a formula is in). The ML also contains the constructors P' and Q' of no arguments, whose denotations are the DBL language formulas P and Q. We can state that the DBL and ML predicates have the same meaning by axioms of the following sort:

$$\begin{aligned} DB(P') &\equiv P \\ DB(Q') &\equiv Q \end{aligned} \qquad P1.$$

$P1$ states that P (or Q) holds just in case the DBL formula represented by P' (or Q')

holds in the database. The axioms $P1$ can be used to derive DBL queries equivalent to any ML query that is a sentential expression over P and Q. Suppose the original ML query is $P \vee Q$. Then it is easy to show that, by $TR1$ and $P1$,

$$DB(or(P', Q')) \equiv P \vee Q \qquad Q1.$$

The process of database query derivation is similar to that of *answer extraction* in formal question-answering systems [6]. The problem of finding a database query is reduced to that of finding a proof of the first-order ML formula that is an instance of $T1$.

6. Predications on Individuals

In the last section we developed an abstract syntax for DBL constant predicates. Since terms referring to individuals are also an important part of the DBL, the abstract syntax must be extended to include constructors for these DBL terms. In this section we introduce the needed technical machinery for this into the ML. Although the machinery may appear cumbersome, the idea behind it is fairly simple: to state the correspondence between terms in the DBL and the individuals they name. The complications arise in keeping track of the distinction between terms in the ML, terms in the OL, and the individuals they denote.

6.1 The Denotation Function

We begin by defining a new sort, DBTS, that is the set of DBL *terms*. ML terms of this sort will denote DBL terms. Again, a primed convention will be used; e.g., the ML constant term A' will denote the DBL constant term A. We can diagram this relationship as follows:

$$\begin{tikzcd}
\text{ML:} & A \arrow[dd] & B \arrow[dd] & A' \arrow[d] & B' \arrow[d] \\
\text{DBL:} & & & A \arrow[dll] & B \arrow[dll] \\
\text{individuals:} & \mathcal{A} & \mathcal{B} & &
\end{tikzcd}$$

The arrows in the diagram represent the denotations of the terms. A and B as terms in the ML refer to the individuals $\mathcal{A}$ and $\mathcal{B}$ in the domain of discourse; they are of the sort INDS. Similarly, the terms A and B of the DBL also denote individuals in the domain. The ML constants A' and B' (of sort DBTS) refer to the DBL terms A and B, rather than to individuals. There is nothing special about using the same symbols in both languages to denote these individuals; it is done here simply for consistency of naming.

To construct DBL expressions that involve predications over arguments, the ML contains a primed constructor symbol P' for each predicate symbol P of the DBL. The arguments of the ML constructor P' are DBL terms, one for each of P's arguments. Thus, the denotation of the constructor P' is a DBL predicate over DBL terms; $P'(A',B')$ denotes the DBL predicate $P(A, B)$.

Although the DBL and the ML can have different domains of individuals, it is technically convenient to make them identical so the correspondence between quantified expressions in the two languages easier to state. To assert that the DBL's model actually has enough individuals, the denotation and naming functions must be introduced into the metalanguage.

It is often useful to know, in the ML, what individual a DBL term refers to. The denotation function Δ is used for this purpose;[6] it takes a DBL term as its argument, and returns the individual denoted by that term. In the ML, the sorts of the argument and result of the denotation function are given by:

$$\forall n/_{\mathrm{DBTS}}\, \exists x/_{\mathrm{INDS}}\ \Delta(n) = x.$$

The Δ function can be understood more easily by examining the denotation map for individuals in the ML and the DBL; it maps between a DBL term and its referent. Because, in the ML, A' refers to the DBL term whose denotation is the same as the ML term A, we write:

$$\Delta(A') = A.$$

6.2 Standard Names

In any language, there may be many different names for the same individual; e.g., we could speak of the Lafayette as "the ship commanded by Jones." Such names might change their denotations in different circumstances, however. It is handy to have a name for each individual that is always guaranteed to refer to that individual; these are called *standard names.* All the constant terms we introduce into the DBL will be such standard names (e.g., *LAFAYETTE* from Section 3), since there is no need to have more than one constant term for the same individual in the DBL.

[6]Church [2] introduced the denotation function to formally describe the denotation of terms in the object language.

In the ML, it will be useful to construct the standard DBL name for individuals. We define the standard name function η of one argument, an individual. The value of η is the standard DBL constant term that refers to that individual. Thus we would write:

$$\forall x/_{\mathrm{INDS}}\, \exists n/_{\mathrm{DBTS}}\, \eta(x) = n$$

and

$$\eta(A) = A'.$$

In the denotation diagram above, η is the mapping from individuals to their standard name in the DBL.

We state here two useful properties of the standard name function:

$$\forall xy/_{\mathrm{INDS}}\, [x \neq y] \equiv DB(not(\eta(x) =' \eta(y))) \qquad DN1,$$

$$\forall x/_{\mathrm{INDS}}\, \Delta(\eta(x)) = x \qquad DN2.$$

The ML function $='$ constructs the DBL equality predicate of its two arguments. $DN1$ says essentially that the two DBL standard names for different individuals actually do refer to different individuals. $DN2$ establishes η as the DBL standard name function for all individuals referred to by the ML, because it guarantees that the denotation of an individual's DBL name is indeed that individual.

6.3 Quantified DBL Expressions

Quantified DBL expressions are constructed with the ML functions *some* and *all*. Both these functions take three arguments: a DBL variable name, a DBL sort, and a DBL expression. We generally use ML constant terms t' and s' to denote DBL tuple variables t and s, and primed constant terms to denote the corresponding sorts in the DBL. Thus:

ML term	denotes	DBL formula
$some(t', \mathrm{SHIPR}', sname'(t') =' FOX')$		$\exists t/_{\mathrm{SHIPR}}\, sname(t) = FOX$
$all(t', \mathrm{SHIPR}', length'(T') >' \eta(344.4))$		$\forall t/_{\mathrm{SHIPR}}\, length(t) > 344.4$

6.4 Corner-Quotes

The abstract syntax can be a cumbersome way of naming complicated DBL expressions within the ML. To reduce the notational burden, we introduce an abbreviation device: we let a DBL expression stand for itself in ML formulas. The DBL formula will be set off with corner-quotes from the rest of the ML formula so there will be no confusion. Below are examples of the use of corner-quotes:

<u>ML abbreviation</u>	stands for	<u>ML formula</u>
$DB(\ulcorner P(A,B)\urcorner)$		$DB(P'(A',B'))$
$DB(\ulcorner P \vee Q\urcorner)$		$DB(or(P',Q'))$
$DB(\ulcorner \forall t/_{\text{SHIPR}}\ sname(t) = FOX\urcorner)$		$DB(all(t', \text{SHIPR}', sname'(t') =' FOX'))$

The translation between corner-quote abbreviations and their corresponding ML terms is straightforward: all predicate and term symbols in the corner-quote expression are replaced by primed symbols, and booleans and quantifiers map back to their constructors in the ML. It should be noted that the corner-quote convention is strictly a device for abbreviating ML terms; it does not introduce any logical machinery into the ML.

ML terms of the form $\eta(x)$ are commonly found in DBL term constructions, where x is a ML variable ranging over individuals (sort INDS). We therefore extend the corner-quote conventions so that if a ML variable of sort INDS occurs within corner-quotes, it translates to the standard name of that variable. So, for example, the denotation axiom $DN1$ would be written as:

$$\forall xy/_{\text{INDS}}\ [x \neq y] \equiv DB(\ulcorner x \neq y\urcorner),$$

where $\ulcorner x \neq y\urcorner$ is an abbreviation for $not(\eta(x) =' \eta(y))$.

In addition, we allow ML variables of the sort DBFS to appear within corner-quote abbreviations; they remain unchanged in translation. The only use of this convention will be in Axiom $P6$ in Section 8.4; there, for example, $\ulcorner \forall t/_{\text{SHIPR}}\ f\urcorner$ is an abbreviation for $all(t', \text{SHIPR}', f)$, where f is a ML variable of sort DBFS.

7. Some Query Derivations

We are now in a position to state the relationship between the ML predicates *DOC*, *COM*, *etc.*, and the relations of the database. Suppose it is the case that the database contains complete information about the commander of every ship, so there is an {*officer,ship*} tuple in the COMMANDR relation for each officer and ship in the domain of discourse. The correspondence between the *COM* predicate and the COMMANDR relation can be stated as:

$$\begin{aligned}&\forall x/_{\text{OFFICERS}}\forall y/_{\text{SHIPS}}\\&\quad[DB(\ulcorner\exists t/_{\text{COMMANDR}}[officer(t)=x\wedge ship(t)=y]\urcorner)\\&\qquad\equiv COM(x,y)]\end{aligned}\qquad P3.$$

Note that because of the corner-quote convention, x and y will be replaced by $\eta(x)$ and $\eta(y)$ in the ML term naming the DBL expression. Axiom $P3$ can be used to derive database queries relating to the *COM* predicate.

Example. The query to be answered is "Does Jones command ship USN123?" By $T1$, this is expressed in the ML as:

$$\exists f/_{\text{DBFS}}\ [DB(f)\equiv COM(JONES, USN123)] \qquad Q2.$$

If the standard DBL names for Jones and USN123 are also *JONES* and *USN123*, then by $P3$ the equivalent DBL expression is:

$$\exists t/_{\text{COMMANDR}}[officer(t)=JONES\wedge ship(t)=USN123].$$

A more complicated correspondence exists between the *DOC* predicate and the SHIPR relation:

$$\forall x/_{\mathrm{SHIPS}}\, DB(\ulcorner \exists t/_{\mathrm{SHIPR}}\, ship(t) = x \wedge medic(t) = T \urcorner)$$
$$\supset \exists y/_{\mathrm{DOCTORS}}\, DOC(y, x)$$
$$\forall x/_{\mathrm{SHIPS}}\, DB(\ulcorner \exists t/_{\mathrm{SHIPR}}\, ship(t) = x \wedge medic(t) = F \urcorner) \qquad P4.$$
$$\supset \sim[\exists y/_{\mathrm{DOCTORS}}\, DOC(y, x)]$$
$$DB(\ulcorner \forall t/_{\mathrm{SHIPR}}\, medic(t) = T \vee medic(t) = F \urcorner)$$

The first two assertions state that if the *MEDIC* element of a SHIPR tuple is T, then there is a doctor on board the ship; and if it is F, then there is not. The last assertion says that the *MEDIC* field will always contain one of T or F.

Example. The query to be answered is "Is there a doctor on board USN123?" From $T1$, the ML expression is:

$$\exists f/_{\mathrm{DBFS}}\, [DB(f) \equiv \exists x/_{\mathrm{DOCTORS}}\, DOC(x, USN123)] \qquad Q3.$$

By using $P4$ above, either of the following two DBL expressions can be proven to satisfy $Q3$:

$$\exists t/_{\mathrm{SHIPR}}\, ship(t) = USN123 \wedge medic(t) = T$$
$$\sim[\exists t/_{\mathrm{SHIPR}}\, ship(t) = USN123 \wedge medic(t) = F]$$

Given just $P4$, it is not possible to find a DBL expression corresponding to the query, "Is Jones the doctor on board USN123?" The domain of discourse represented by the *DOC* predicate properly subsumes the database's information on the subject. Thus, this query can be represented in the ML, but not in the DBL.

8. Incomplete Information

In the previous section we encountered an example of a database that had incomplete information about the domain of discourse: there was no representation for which doctor was on board a ship, only an indication that some doctor was present. In this section, we examine a few of the more common ways in which a database may have incomplete information about the domain. We show how to encode all of the types of partial information that characterized the representational inadequacy of the database in Section 2.

8.1 Absence of a Relation

It often occurs that some part of the domain of discourse is not referred to at all by the database. For the sake of conceptual completeness, for example, we may allow the user to ask about properties of ships that are not included in the database under consideration. In this case, there will be a predicate in the ML that refers to the property, but no corresponding DBL tuple set. This case is easy to encode: there simply is no axiom that relates the predicate to any relation in the database.

8.2 Conceptual Mismatch

This type of incompleteness occurs when a ML predicate has only a partial correspondence in some database relation. An example of this is the *DOC* predicate in the previous section (Axiom *P4*); only ML queries involving existential quantification over the doctor argument could be answered. Partial correspondence of a ML predicate is characterized by the ability to find DBL expressions for some, but not all, of the ML queries containing the predicate. Complicated relationships between ML predicates and DBL tuple sets can be encoded because the ML has the expressiveness of a full first-order language.

8.3 Only-If Incompleteness

The correspondence previously stated in Axiom *P*3 between the *COM* predicate and the COMMANDR relation used an equivalence connective. The only-if half of this connective is important for the derivation of DBL expressions equivalent to ML formulas involving *COM*. This is because the only-if half of $P3$ states that if the tuple $\{A, B\}$ is not present in the COMMANDR relation, then A does not command B.

In many cases, however, the interpretation of a database relation is that if a tuple is present, then the relationship holds between those individuals; but if it is not present, it cannot be inferred that the relationship does not hold. All the positive instances of the relation correspond with the domain of discourse, but the negative instances do not. For example, it may be the case that all the commanders of U.S. naval ships are known, but not those of foreign naval ships. Then Axiom $P3$ is too strong, and must be weakened to an implication:

$$\begin{aligned}&\forall x/\text{OFFICERS}\,\forall y/\text{SHIPS}\\ &\quad[DB(\ulcorner \exists t/\text{COMMANDR}\,[\mathit{officer}(t) = x \wedge \mathit{ship}(t) = y]\urcorner)\\ &\qquad\supset COM(x, y)]\end{aligned} \qquad P3'.$$

This axiom is not strong enough to allow the derivation of equivalent DBL expressions for the *COM* predicate, but may still be useful in an intelligent question-answering system. Failing to find a proof of an instance of $T1$ for a ML query, such a system might weaken $T1$ to find a DBL formula that implies, but is not implied by, the original query. If a positive answer is returned from the database, then the original query is true; if not, no conclusion can be drawn.

8.4 Domain Incompleteness

Since the DBL is a tuple calculus, there is no explicit quantification over subdomains of INDS, e.g., SHIPS or OFFICERS. How then can DBL queries be formed that ask whether a property holds for all the members of one of these sets? For example, consider the ML query:

$$\forall x/\text{SHIPS}\,\exists y/\text{LENGTHS}\,[LEN(y, x) \wedge (y > 344.4)] \qquad Q5.$$

This asks whether all ships have a length greater than 344.4 meters. Suppose we assume that every ship length is represented in the SHIPR relation:

$$\begin{aligned}&\forall x/_{\text{SHIPS}}\,\forall y/_{\text{LENGTHS}}\\ &\qquad[DB(\ulcorner\exists t/_{\text{SHIPR}}\, \mathit{length}(t) = y \land \mathit{ship}(t) = x\urcorner)\\ &\qquad\qquad\equiv LEN(y, x)]\end{aligned} \qquad P5.$$

$P5$ is not sufficient to answer $Q5$; the reason is that we have not stated anything about whether all ships are included in the SHIPR relation or not. A counterexample to the truth of $Q5$ would be a domain where there were some ships that did not have lengths. The best we can do with $P5$ is to derive the equivalent ML formula:

$$\forall x/_{\text{SHIPS}}\, DB(\ulcorner\exists t/_{\text{SHIPR}}\, \mathit{length}(t) > 344.4 \land \mathit{ship}(t) = x\urcorner) \qquad Q6,$$

where there is still a ML quantifier over all ships.

Incompleteness of any domain is thus automatically assumed unless explicit nonlogical axioms are included to counteract it. For the ships world we are using as an example, suppose we want to say that all ships in the domain are to be found in the SHIPR relation, but not all LENGTHS or SNAMES need be present. That is, the domain LENGTHS is assumed to be the rational numbers in the interval (say) $[100, 1000]$; at any given moment, only a finite subset of these will be present in the SHIPR relation. Similarly, more SNAMES are available than are in use.

To state the completeness of the SHIPR relation with respect to SHIPS, we assert:

$$\begin{aligned}&\forall f/_{\text{DBFS}}\,[\forall x/_{\text{SHIPS}}\, DB(\ulcorner\exists t/_{\text{SHIPR}}\, \mathit{ship}(t) = x \land f\urcorner)]\\ &\qquad\equiv DB(\ulcorner\forall t/_{\text{SHIPR}}\, f\urcorner)\end{aligned} \qquad P6.$$

That is, a universal quantifier over SHIPS can be moved inside the DB predicate to the SHIPR relation. The use of $P6$ enables us to prove that the following DBL expression satisfies $Q5$:

$$\forall t/_{\text{SHIPR}}\, \mathit{length}(t) > 344.4.$$

On the other hand, because there is no explicit domain completeness axiom like $P6$ for LENGTHS, the question "Is there some ship of every length?" cannot be answered by a DBL query.

The ability to correctly represent that the database has only partial information about a domain of individuals enables us to allow infinite domains in the database, without deriving DBL expressions that yield counterintuitive truth-values for ML queries. If answering a ML query involves quantifying over an infinite domain, no equivalent DBL expression will ever be generated for it.

9. Other Issues

In this section we briefly describe how open queries can be accommodated in the metalanguage approach. We also examine several ways in which a question-answering system might use the ML/DBL representation to respond intelligently to user queries.

9.1 Open Queries

An open query is a query whose answer is a set of individuals, rather than a truth-value. In the metalanguage, open queries can be represented by a formula that has one or more free variables in it. The answer to the query is the set of those individuals that, when substituted for the free variables, make the query true.[7] For simplicity, we consider open formulas with only one free variable, which we indicate by enclosing it in square brackets next to the ML formula: $M[x]$ is a ML formula with free variable x. Free variables in the ML will always be of the sort INDS.

In the DBL, we allow free variables over the simple (nontuple) domains. Because DBL tuple domains are finite, DBL expressions involving free variables have the important property that only a finite set of individuals will satisfy the expression when substituted for the free variable.[8] A DBL query with a free variable is thus guaranteed to return a finite set of individuals; moreover, because of the computational properties of the DBL discussed in Section 3, an algorithm exists for determining this set.

The problem of finding a DBL open formula that corresponds to a ML open formula can be stated in terms similar to $T1$, the schema for closed queries:

$$\forall x/_{\text{INDS}}\, \exists f/_{\text{DBFS}}\, (DB(f[\alpha/\eta(x)]) \equiv M[x]) \qquad T2.$$

Here $f[\alpha/\eta(x)]$ is the DBL expression constructed by substituting the term $\eta(x)$ for the

[7]Reiter [15] extends the notion of the answer to an open query to disjunctive combinations of individuals, e.g., "either Smith or Jones commands the Lafayette." We do not consider this complication here.

[8]To show this, it is necessary to exclude from the DBL expressions of the form $\alpha = \alpha$, where α is a free variable.

free DBL variable α in f. If a constructive proof of $T2$ can be found, then it will yield an open DBL formula whose truth-value is the same as that of $M[x]$ for every individual x. This formula can then be evaluated against the database to produce the required set of individuals.

9.2 Types of Questions and Answers

The ability to formalize the relationship of the database to the domain opens up new possibilities for intelligent response to a user's queries. One type of query that can be readily handled in this framework is a request concerning what information the database has about the domain. For example, suppose a naive user wants to know if the database has information about what doctors are on board which ships. This question could be phrased in the ML as:

$$\forall x/\text{SHIPS}\ \forall y/\text{DOCTORS}\ [\exists f/\text{DBFS}\ DB(f) \equiv DOC(x,y)] \qquad Q7.$$

If this ML formula could be proven, then any query about individual doctors and ships could be answered. Note that we are not interested in evaluating the DBL formula that is a binding for f in this case; rather, we wish to establish the existence of a class of DBL formulas.

In a more speculative vein, we might consider integrating the ML/DBL framework with current AI formalisms to represent changing states of the world– most notably the situation calculus [13]. Suppose each ML predicate were extended to take an additional argument, a ***situation*** in which the predicate was to hold. Thus $DOC(x,y,s)$ would mean that x was the doctor on board y in situation s. There would be some distinguished situation S_0, the current state of the world, about which the database would have information. It would then be possible to correctly represent and answer user queries that made the distinction between a proposition always being true of the domain, as opposed to true in the current situation. For example, consider the two queries:

Is there always a doctor on every ship?

Is there a doctor on every ship now?

The first can be answered only by proving a ML formula over all situations; the second can be answered by consulting the current state of the database.

If several previous copies of the database are retained, then it is also possible to answer queries about past situations. Suppose, for example, that the day associated with each situation is kept in the ML, and a separate copy of the daily database is available for querying. Then queries such as "Do more ships have doctors on board today than yesterday?" could be answered by evaluating two database queries and computing the answer from the values they returned.

9.3 The User's View

While this paper has drawn upon the particular application area of database access for concreteness, the metalanguage/object language approach can be presented in a more general setting. Many complicated systems, especially Artificial Intelligence systems, have to interact with a user (which may perhaps even be another system). The user's view of the domain of discourse might be different from that of the system; generally, the user will not have as much expertise and may not even talk the "language" that the system uses. Within such a situation, we might rephrase the problem given in Section 2.2 as:

> How do we keep the complicated system as a repository of knowledge about the domain of discourse, while at the same time reconciling its view of the world with that of its users?

To make such systems useful thus requires a sophisticated representation of both the system and the user of the system, a problem that has been central to this paper. It would be interesting to see if the formalism presented here could help to give a more rigorous foundation to the previous work on this problem, *e.g.*, the MACSYMA assistant [5] and, more recently, the CONSUL system [11].

10. Relation to Other Work

Recent work in formalisms for database representation, which has been collected in [4], differs broadly from the approach presented here in that the database is viewed as a set of ground atomic sentences in a first-order theory of the domain of discourse, rather than as a model. In terms of the framework given in Section 2.2, the "more powerful representation" is a first-order language, while the database is a collection of ground atomic formulas in that language. A query about the domain of discourse is initially phrased in the first-order language; to generate an equivalent query against the database, the query must be transformed into a form that can be answered by looking only at the ground atomic formulas in the database. The way this is accomplished is that a set of predicates (the *database predicates*) are set aside for use in the database part of the language, and axioms specify the relationship of these database predicates to other predicates in the language. The transformation of the query is then effected by replacing all nondatabase predicates with database ones, based on the axioms. We might call this the *transformational* approach.

Unfortunately, the transformation of a query is from one formula in the language to another, based on syntactic criteria. It is impossible to formalize this transformation within the language itself, so special procedures must be developed to implement the transformation. The procedures that have been given in the literature, so as to be proven complete, place heavy restrictions on the expressiveness of the first-order language, both in terms of the axioms that relate nondatabase predicates to database ones, and the assumptions that must be made about incomplete information.

The assumptions about incomplete information that are made have been called the *Closed World Assumption* and *Domain Closure.* The closed world assumption says that a relationship that cannot be proven to exist actually does not exist. Making the closed world assumption is similar to assuming that there is no only-if and domain incompleteness for any predicate of the language, although, of course, only-if and domain incompleteness were stated with respect to an interpreted language and not an uninterpreted theory as in the transformational approach. Domain closure is the assumption that there is a constant symbol for every individual in the language (and usually requires that constants be standardized apart).

Unfortunately, these two rather strict assumptions do not always correspond with our intuitive notions about incomplete information in the database, and do not give the required flexibility in axiomatizing the relationship of the database to the domain. For example, there is no way to state that some database predicates are only-if complete with respect to the domain of discourse, while others are incomplete.

In the transformational approach, the representational power of the first-order language used to describe the domain is severely restricted so as to carry out database query derivation in the presence of the incompleteness assumptions. In particular, these systems forego the use of existential quantification and function symbols in axioms that connect the database and nondatabase predicates. This means, for example, that a relationship such as Axiom $P4$ could not be encoded.

The systems described in [4] suffer from these limitations to a greater or lesser extent, depending on which trade-off they choose to make. For example, Chang [1] chooses a very restrictive form for his axioms, but has a simple algorithm for deriving equivalent formulas that involve only database relations. Reiter [15] relaxes these restrictions somewhat, and also gives an exact account of the assumptions being made about incomplete information; but his derivational algorithm is more complicated.

It should be mentioned that the criticisms leveled above apply to the so-called *evaluational* framework of database query derivation, which is assumed in this paper. That is, the database is presented *a priori*, and must be addressed by a separate query processor whose communication overhead is high relative to deduction outside the database. With this assumption, it is reasonable to try to transform the full query to one that can be evaluated all at once against the database. Other systems, such as [8], assume a much tighter coupling between the database and the nondatabase parts of the first-order theory. The database is assumed to be simply a repository for atomic ground formulas that can be accessed during a proof, although some attempt is made to minimize the number of accesses.

11. Conclusion

We have taken the view that for the purposes of question-answering a relational database can best be represented as a model of a particular type of first-order language, a tuple relational calculus. This view has proved particularly useful in two respects. First, by axiomatizing the database language and its associated model in a metatheory, we have been able to describe in a powerful and flexible manner how the database corresponds to the domain of discourse. This facility is a representational advance, because AI systems that must address databases need just this facility. Secondly, viewing the database as a mechanizable model of the DBL enables us to take advantage of the computational properties of database query language processors. Once a database query that is equivalent to an original query is derived, it can be evaluated against the database to determine the truth of the original query. Thus the algebraic operations of the database processor can be incorporated in an elegant way into the deductive process of question-answering.

A final word about implementation. An initial algorithm that incorporated some of the ideas about incomplete information in the database was implemented for the D-LADDER project [9]. A restricted first-order language (the *conceptual schema*) was used to represent the domain of discourse, and a query expressed in this language was transformed by the algorithm into the database language SODA [14]. This algorithm, however, did not take advantage of the full power of the metalanguage encoding. In KLAUS, an intelligent knowledge-acquisition and question-answering system, we intend to implement the ML/DBL structure as described in this paper, and explore the complicated issues of deduction that will arise.

References

[1] Chang, C. L., "DEDUCE 2: Further Investigations of Deduction in Relational Data Bases," in *Logic and Data Bases* , H. Gallaire and Jack Minker (Eds.), Plenum Press, New York (1978).

[2] Church, A., "A Formulation of the Logic of Sense and Denotation," in *Structure, Method, and Meaning* , P. Henle *et. al.* (Eds.), Liberal Arts Press, New York (1951).

[3] Codd, E. F., "Relational Completeness of Data Base Sublanguages," in *Data Base Systems* , R. Rustin (Ed.), Prentice-Hall, Englewood Cliffs, N.J. (1972).

[4] Gallaire, H. *et. al.*, "An Overview and Introduction to Logic and Data Bases," in *Logic and Data Bases* , H. Gallaire and Jack Minker (Eds.), Plenum Press, New York (1978).

[5] Genesereth, M. C., "The Role of Plans in Intelligent Teaching Systems," in *Intelligent Teaching Systems* , D. Sleeman (Ed.) (1980).

[6] Green, C. C., "Theorem Proving by Resolution as a Basis for Question Answering Systems," in *Machine Intelligence 4* , B. Meltzer and D. Michie (Eds.), American Elsevier, New York (1969).

[7] Hendrix, G. G. *et. al.*, "Developing a Natural Language Interface to Complex Data," *ACM Transactions on Database Systems* **3**, 2 (June 1978).

[8] Kellogg, C. *et. al.*, "Deductive Planning and Pathfinding for Relational Data Bases," in *Logic and Data Bases* , H. Gallaire and Jack Minker (Eds.), Plenum Press, New York (1978).

[9] Konolige, K., "A Framework for a Portable Natural-Language Interface to Large Data Bases," *Artificial Intelligence Center Technical Note 197* , SRI International, Menlo Park, California (October 1979).

[10] Levesque, H. J., "The Interaction with Incomplete Knowledge Bases: A Formal Treatment," *IJCAI-7* , Vancouver, B.C. (1981), pp. 240–245.

[11] Mark, W., "Representation and Inference in the Consul System," *IJCAI-7* , Vancouver, B.C. (1981), pp. 375–381.

[12] McCarthy, J., "Towards a Mathematical Science of Computation," *Information Processing, Proceedings of the IFIP Congress* **62**, North-Holland, Amsterdam (1962), pp. 21–28.

[13] McCarthy, J. and Hayes, P. J., "Some Philosophical Problems from the Standpoint of Artificial Intelligence," in *Machine Intelligence 9* , B. Meltzer and D. Michie (Eds.), Edinburgh University Press, Edinburgh (1969), pp. 463–502.

[14] Moore, R. C., "Handling Complex Queries in a Distributed Data Base," *Artificial Intelligence Center Technical Note 170* , SRI International, Menlo Park, California (October 1979).

[15] Reiter, R., "Deductive Question-Answering on Relational Data Bases," in *Logic and Data Bases* , H. Gallaire and Jack Minker (Eds.), Plenum Press, New York (1978).

[16] Ullman, J. D., *Principles of Database Systems,* Computer Science Press, Potomac,

Maryland, 1980.

[17] Weyhrauch, R., "Prolegomena to a Theory of Mechanized Formal Reasoning," *Artificial Intelligence* **13** (1980).

[18] Woods, W. A. *et. al.*, "The Lunar Sciences Natural Language Information System," *BBN Report 2378* , Bolt Beranek and Newman, Cambridge, Massachusetts (1972).