A METALANGUAGE REPRESENTATION OF RELATIONAL DATABASES FOR DEDUCTIVE QUESTION-ANSWERING SYSTEMS

Kurt Konoligc

SRI International
Mcnlo Park, Ca.

ABSTRACT

This paper[1] presents a method of formally representing the information that exists in a relational database. The primary utility of such a representation is for 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 database. The problem that then arises is 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. This view proves particularly useful in two respects. First, by axiomatiiing the database language and its associated model in a metatheory, we are able to describe in a powerful and flexible manner how the database corresponds to the domain of discourse. Secondly, viewing the database as a mechanizable model of the database language 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.

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 AI, but there is a need to reconcile it with AI representations. Many AI systems that address real-world domains must access existing databases; typicallly these have been natural-language question-answering systems, e.g., LUNAR [14] and D-LADDEIi (8). The D-LADDER database, for example, is a pre-existing relational database that is distributed across several sites of the Arpanet; access to the information is through a pre-defined 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 or not there is a database query that will answer the original question.[2] Other researchers, including Rciter (12) and Chang [1], have advanced solutions to the problem posed above; but their solutions are inadequate for a number of reasons. A common fault is that they compromise the expressive power with which they represent the domain of discourse in order to arrive at a satisfactory algorithm for deriving database queries. We will critique these systems more fully at the end of this paper.

The key task we accomplish in this paper is the formulation of a representation rich enough to describe the domain of discourse, along with information that the associated database contains about that domain. Furthermore, we present an algorithm that will determine when sufficient information exists in the database to decide the truth of some statement about the domain, and generate the appropriate database language query.

To formalise the information content of a relational database, we take the view that the database is a finite *model of* a particular first-order language, called the *database language,* or DDL. The operations of the relational calculus can then be used in a decision procedure for sentences in the language. When this occurs, the database language is said to be interpretable, with the database as its intended model. Queries to the database are expressed as sentences in the DHL; operations on the relations of the database can be used to find the truth-value of any formula of the DDL. A model with this property is said to be a *computable model.*

When viewed as an interpreted language, the DDL lacks the descriptive power necessary for most AI applications, e.g., it is impossible to express disjunctive facts ("cither it is raining or the sun is out"). Thus we will axiomalizc the 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. In addition to encoding the domain of discourse, the ML theory will contain a description of the DDL. That is, the ML theory will have terms that denote DDL *expressions,* and predicates that assert the truth of DDL expressions in their intended model, the database. Finally, the ML theory will contain axioms that relate the truth of sentences in the DDL to predicates in the ML that describe the domain of discourse. These axioms characterize the connec-

[1]The work reported here was supported by the Defense Advanced Research Project* Agency und«r Contract N00039-80 r 0645. The views and conclusions contained in this document arc thoso of the authors and should not be interpreted as representative of th« official policies of the Defense Advanced Research Projects Agency of the U.S. Government.

[2]Note that we are not making a judgement here about the utility of database representation technology. Il may be the case that all the information that is present in the database under consideration could be more easily and conveniently represented using standard AI techniques, e.g., semantic nets or Morn clauses. The situations we are interested to are those where the database is a pre-defined part of the domain, and wt must build some representation for the information that 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.

tion 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 find a DHL expression that is equivalent to the original ML query. In the latter case, because the DDL 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 [5].

The metalanguage/database language dichotomy is useful in making a theoretically satisfying distinction between the description of the discourse domain required by an AJ system, and the information that may be present in an associated database. As a result, it will be possible for a question-answering system that uses this representation to respond in a more sophisticated way to user inquiries to a database. For example, it is possible to correctly represent the distinction between questions such as "Is P always true?" *versus* "Is P currently true in the database?"

2. The Relational Database as a Model

We seek a method of describing the information that a database contains, that is, a way of formally characterizing the way in which data in the database relates to the domain under consideration. For the purposes of this paper, we will 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.[3] A relational database is a set of relations {/?,} over a set of domains *{Dj}.* Each relation *Ft* consists of a set of *tuples* of some fixed length n, the *arity* of the relation; we will indicate a relation's arity with a superscript, R^n. The elements of the tuples are drawn from the domains of the database.[4] In a relational database, the relations arc 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.

As an example relational database, consider the following relations about the ships world:

SHIPR : SHIPS X *SNAMES* X *LENGTHS* X *MEDIC*
CO MM AN DR : OFFICERS X *SHIPS*

The domains are *SHIPS, S NAMES* (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 (6). A typical tuple of the *SHIPR* relation might be *{U S N m,L AF AY ETTE,344.&,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 *{USNl2Z,LAFAYETTEM**,T)* 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.*

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 or not 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 and the domain.

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 in a convenient manner. For example, suppose we wished 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,[5] restrict the resulting relation so that only tuples with a *LENGTH* element greater than *L* remain, and then project the *OFFICER* elements to yield a set of one-element tuples containing the answer.

Formally, the results of algebraic manipulations on a relational database can be described by designing an appropriate first-order language for the database. 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 wc actually have the database in hand, it is possible to determine the .truth value for any expression of the DBL 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 the truth-value of any expression in an appropriately defined DBL; hence the database and its associated algebra forms 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 above. We will use a many-sorted first-order language with equality. There is one sort for each relation in the database; a sort consists of all tuples that are in its corresponding relation. Variables are restricted to range over a given sort. In quantified expressions, the sort that a variable is restricted to will always be indicated by giving the name of the relation for the sort. For example, in

VtlSHIPR • • •

the variable *t* is restricted to tuples in the relation *SHIPR.* Because variables only refer to tuples in a relation, this type of language has been called a *tuple relational calculus* (13).

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,

*Thc approach deicribed here will actually work with any databaie representation technology that is powerful enough to be a mechanizable model for tome first-order language.

[4]This is unfortunate terminology, since 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 referrent of *domain* in this paper, because of context.

*This operation if called an $equijoin_t$ and is the composition of a join with an equality restriction.

we will use function names that are similar to the domain of the element that they select from the tuple. As an example, consider:

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

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 some arithmetic predicates such as greater than and lest 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 [13]; a safe language is one whose expressions can all be interpreted by examining just the instances of relations present in the database. The practical import of this is that safe languages are mechanisable, 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.

3. The Metalanguage

A language like the DDL for which the intended model is available is called an *interpreted language.* If the database correctly reflects the structure of the domain of discourse, then 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 one 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 they 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 *COM MANOR* relation of the form {nnn,£/5JV123}, where 1/5JV123 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.[1]

If a question-answering system is to return correct responses to 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 wish to make use of the information that is contained in the database in those cases where it is sufficient for responding to a query. So the representation we seek must not only characterise the domain of discourse, it must also encode the way in which the database as a model corresponds to the actual world.

To represent the domain of discourse, we will 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:

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

which says that every ship has a length. It is not critical that we have chosen this particular form of the first-order predicate calculus; any of the non-sorted 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 tuples.

In addition to representing the domain of discourse, the ML is also used to characterize the database as a model of the DBL; this is what makes it a metalanguage. In the ML, we will 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 / denotes the DBL formula *Vt/sHtPR* |«name(*) = *LAFAYETTE)* V *length(t)* > 344.6). Then the ML expression *DB(J)* asserts that the DBL formula denoted by / is actually true of the database; that is, all tuples in the *SHI PR* 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 DDL formulas as arguments and return the obvious DBL compound boolean formula. For example, the ML term *and(f, g)* denotes the DBL formula *FAG,* where *F* and *G* are the DBL formulas denoted by / and *g.* This *abstract syntax* for object-language formulas was introduced by McCarthy (9).

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:

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

Additionally, to construct an arbitrary DBL formula we will 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 PA~*Q*. The two ML terms *TRUE'* and FALSE' are specially defined to refer to the DBL propositions *TRUE* and *FALSE,* whose truth values in

•We are going to uiume in this paper that what information the database does contain about the domain of ditcottrac is correct. In principle, the formalisation presented in this taction could readily handle case* where the database was not in conformity with the domain, e.g., the tuple *JONES, USN123* is present in the *COM MANOR* 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.

the database arc always taken to be true and false, respectively:

$$DB(TRUE')=PV\sim P$$
$$DB(FALSE') \text{ s } P\,A\sim P$$

4. Database Query Derivation

Having described the DBL and its associated database within the ML, we are in a position to formulate the derivation of database queries in the DDL from an original ML query. At this point we will 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:

$$3f/DBrs\backslash DB(f)\sim qwff \qquad T\backslash.$$

If a constructive proof of *T* can be found for a given instance of *qwff,* the binding for / will be a DBL formula that is 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 *TV,* if the answer to the DBL language query turns out to be "false," then the negation of qw/f will have been established.

A special case of *T* occurs if the binding for / 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 DHL query.

Example. Let *P* and *Q* be two DBL propositions which we intend to have the same meaning as the ML predicates *P* and *Q,* respectively (wc can use the same names because we will 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:

$$DB(F)\sim P$$
$$DB(Q>)=Q$$

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 *PVQ.* Then it is easy to show that, by *TR1* and P1,

$$DB(OT(P',Q')) = PVQ \qquad Ql.$$

The process of database query derivation is similar to that of *answer extraction* in formal question-answering systems (5|. 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 TI.

5. Predications on Individuals

In section 3 an abstract syntax was developed for DBL constant predicates. But since an important part of the DBL are terms referring to individuals, 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 itself 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.

5.1 The Denotation Function

We begin by defining a new sort, *DBTS,* that is the set of DBL *terms.* ML terms or 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:

The arrows in the diagram represent the denotations of the terms. *A* and *B* as terms in the ML refer to the individuals *A* and *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 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 expression 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 $P(A_tB)$.

Although it is possible for the DBL and the ML to have different domains of individuals, it is technically convenient to make them identical. This will make 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 A is used for this purpose;[7] 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:

$$Vn/_{DBTS} \quad 3\,J/\,/_N\,D\,S \quad A(n) \quad * \quad *\bullet$$

The A 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 referrent. Because, in the ML, *A'* refers to the DBL term whose denotation is the same as the ML term *A,* we write:

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

5.2 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 ' v 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 of the constant terms that we will introduce into the DBL will be such standard names (e.g., *I*AFAYETTE* from section 2),

[7] Church (2) introduced the denotation function to formally describe the denotation of terms in the object language-

since there it no need to have more than one constant term for the same individual in the DBL.

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

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

and

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

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

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

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

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

The ML function $='$ constructs the DBL equality predicate of its two arguments. *DNX* essentially says that the two DBL standard names for different individuals actually do refer to different individuals. *DN2* establishes 17 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.

5.3 Quantified DBL Expressions

Quantified DBL expression 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 will 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
$some(t', SHIPR', sname'(t') =' FOX')$		$\exists t/_{SHIPR}\ sname(t) = FOX$
$all(t', SHIPR', length'(T') >' \eta(344.4))$		$\forall t/_{SHIPR}\ length(t) > 344.4$

5.4 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 will 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 that there is no confusion. Here are some examples of the use of corner-quotes:

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

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 abbreviation of MI. terms; it does not introduce any logical machinery into the ML.

It commonly occurs that ML terms of the form $\eta(x)$ are 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:

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

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 $P6$ in section 7.2; there, for example, $\ulcorner \forall t/_{SHIPR} f \urcorner$ is an abbreviation for $all(t', SHIPR', f)$, where f is a ML variable of sort $DBFS$.

6. 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 that 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:

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

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. $P3$ 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:

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

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

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

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

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

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 assertions 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:

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

By using *PA* above, either of the following two DBL expression can be proven to satisfy Q3:

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

Given just *PA,* 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 it is possible to represent this query in the ML, but not in the DBL.

7. Incomplete Information

In the previous section, we encountered an example of a database which had incomplete information about the domain of discourse: there was no representation for which doctor was on board a ship, just an indication that some doctor was present. In this section, we will examine a few of the more common ways in which a database may have incomplete information about the domain. This catalog does not exhaust the possiblities available for characterizing the relationship of a database to its domain within the MI.; but it does show how to deal with a few of the more common situations that arise in practice.

7.1 No Attachment for a Predicate

It often occurs that some part of the domain of discourse is just 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. 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.

A special case of 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 *PA);* 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.

7.2 Only-if Incompleteness

The correspondence previously stated between the *COM* predicate and the *COMMANDR* relation, expressed in P3, 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 F3 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 of the positive instances of the relation correspond with the domain of discourse, but not the negative instances. For example, it may be the case that all of the commanders of U.S. naval ships are known, but not those of foreign naval ships. Then the axiom P3 is too strong, and must be weakened to an implication:

$$\forall x/_{OFFS}\, \forall y/_{SHIPS}\, [DB(\ulcorner \exists t/_{COMMANDR}\, [officer(t) = x \wedge ship(t) = y]\urcorner) \supset COM(x,y)]$$

This axiom it 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 *T* 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.

7.3 Domain Incompleteness

Since the DBL is a tuple calculus, there is no explicit quantification over subdomains of *INDS,* e.g., *SHIPS, OFFICERS, etc.* 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:

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

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:

$$\forall x/_{SHIPS}\, \forall y/_{LENGTHS}\, [DB(\ulcorner \exists t/_{SHIPR}\, length(t) = y \wedge ship(t) = x\urcorner) \equiv LEN(y,x)] \qquad P5.$$

PS is not sufficient to answer *QS;* 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 *QS* would be a domain where there were some ships that did not have lengths. The best we can do with *PS* is to derive the equivalent ML formula:

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

where there is still a ML quantifier over all ships.

Domain incompleteness is thus automatically assumed unless explicit non-logical axioms are included to counteract it. For the ships world we are using as an example, we want to say that all ships in the domain are to be found in the *SHIPR* relation; on the other hand, not all *LENGTHS* or *NAMES* 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 *NAMES* are available than are in use.

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

$$\forall f/_{DBFS}\, [\forall x/_{SHIPS}\, DB(\ulcorner \exists t/_{SHIPR}\, ship(t) = x \wedge f\urcorner)] \equiv DB(\ulcorner \forall t/_{SHIPR}\, f\urcorner) \qquad P6.$$

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

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

On the other hand, because there is no explicit domain completeness axiom like *PS* 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 DDL expressions that yield counterintuitive truth values for ML queries. If answering a ML query involves quantifying over an infinite domain, then no equivalent DBL expression will ever be generated for it.

8. Other Issues

In this section we will briefly describe how open queries can be accomodated in the metalanguage approach. We will also examine some of the ways in which a question-answering system might use the representation to respond intelligently to user queries.

8.1 Open Queries

Open queries are queries 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.[8] For simplicity, we will consider open formulas with only one free variable. The free variable will be indicated by enclosing it in square brackets next to the ML formula: *M\x* is a ML formula with free variable z. Free variables in the ML will always be of the sort *INDS*.

In the DBL, we will allow free variables over the simple (non-tuple) domains. Because of the finitencss of the DBL tuple domains, 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.[9] A DBL query with a free variable is thus guaranteed to return a finite set of individuals; further, because of the computational properties of the DBL discussed in the second section, there exists an algorithm 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 T I, the schema for closed queries:

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

Here $f[\alpha/\eta(z)]$ is the DBL expression constructed by substituting the term *TJ(Z)* for the free DBL variable or in /. 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\z* for every individual z.

8.2 Types of Questions and Answers

The ability to formalise 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:

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

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 / 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/DDL framework with current AJ formalisms for the representation of changing states of the world, most notably the situation calculus (10). 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 z was the doctor on board *y* in situation *a.* There would be some distinguished situation 50, 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 of these can only be answered 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 kept around, 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 MI,, 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. Relation to Other Work

Some recent work in formalisms for database representation has been collected in |4j. This work differs broadly from the approach presented here in that the dabatase is viewed as a set of ground atomic sentences in a first-order theory of the domain of discourse, rather than as a model. There arc several disadvantages to interpreting the database as part of the syntactic description of the domain, which the approach described here does not suffer from:

- e Because the database tuples are taken to be part of the language that describes the domain, special algorithms must be formulated to translate from a formula containing non-database predicates to an equivalent one containing just database predicates,
- t Special assumptions about incomplete information must be made, because it cannot be assumed that the negation of a tuple not present in a relation holds. These special assumptions, which have been called the *Closed World Assumption* and *Domain Closure,* 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.
- The representational power of the first-order language used to describe the domain is severely restricted in order to carry out database query derivation in the presence of the incompleteness assumptions. In particular, these systems forgo the use of existential quantification and function symbols in axioms that connect the database and non-database predicates. This means, for example, that a relationship

'Reiter (12) 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.

•To show this, it is necessary to exclude from the DBL expressions of the form a = Q, where a free variable.

such as *Pi* could not be encoded.

The systems described in [◁] suffer from these limitations to a greater or lesser extent, depending on which trade-off they choose to make. For example, Chang [l] chooses a very restrictive form for his axioms, but has a simple algorithm for deriving equivalent formulas that involve only database relations. Reiter [12] 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* approach to 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 [7], assume a much tighter coupling between the database and the non-database parts of the first-order theory. The database is assumed to be simply a repository for atomic ground formulas that can be accessed during the course of a proof, although some attempt is made to minimize the total number of accesses.

10. Conclusion

We have taken the view that the information content of 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 axiomatiding the database language and its associated model in a metathcory, we have been able to describe in a powerful and flexible manner how the database corresponds to the domain of discourse. This 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 [8|. 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 |11|. However, this algorithm did not take advantage of the 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. at.* (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. at.,* "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) 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).

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

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

(8) 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).

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

(10) 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.

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

[12] 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).

[13] Ullman, J. D., *Principles of Database Systems,* Computer Science Press, Potomac, Maryland, 1980.

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