DEDUCTIVE QUESTION-ANSWERING ON RELATIONAL DATA BASES

by

RAYMOND REITER

TECHNICAL REPORT 77-15

1977 OCTOBER

DEPARTMENT OF COMPUTER SCIENCE
THE UNIVERSITY OF BRITISH COLUMBIA
VANCOUVER, BRITISH COLUMBIA V6T 1W5

ABSTRACT

The principal concern of this paper is the design of a retrieval system which combines current techniques for query evaluation on relational data bases with a deductive component in such a way that the interface between the two is both clean and natural. The result is an approach to deductive retrieval which appears to be feasible for data bases with very large extensions (i.e. specific facts) and comparatively small intensions (i.e. general facts). More specifically, a suitably designed theorem prover "sweeps through" the intensional data base, extracting all information relevant to a given query. This theorem prover never looks at the extensional data base. The end result of this sweep is a set of queries, each of which is extensionally evaluated. The union of answers returned from each of these queries is the set of answers to the original query.

One consequence of this decomposition into an intensional and extensional processor is that the latter may be realized by a conventional data base management system. Another is that the intensional data base can be compiled using a theorem prover as a once-only compiler.

This paper is essentially an impressionistic survey of some results which are rigorously treated elsewhere. As such, no proofs are given for the theorems stated, and the basic system design is illustrated by means of an extended example.

Presented at the Workshop on Logic and Data Bases, Toulouse, France, November 16-18, 1977.

This paper was written with the financial support of the National Research Council of Canada under grant A7642.

DEDUCTIVE QUESTION-ANSWERING ON RELATIONAL DATA BASES

by

Raymond Reiter
Department of Computer Science
University of British Columbia
Vancouver, B.C., Canada

1. INTRODUCTION

The principal concern of this paper is the design of a retrieval system which combines current techniques for query evaluation on relational data bases [e.g. Codd 1972] with a deductive component in such a way that the interface between the two is both clean and natural. The result is an approach to deductive retrieval which appears to be feasible for data bases with very large extensions (i.e. specific facts) and comparatively small intensions (i.e. general facts). More specifically, a suitably designed theorem prover "sweeps through" the intensional data base, extracting all information relevant to a given query. In particular, this theorem prover never looks at the extensional data base. The end result of this sweep is a set of queries, each of which is extensionally evaluated. The union of answers returned from each of these queries is the set of answers to the original query.

There are two important consequences of this decomposition of the question-answering task into a theorem prover computing on the intensional data base, and an extensional processor computing on the extensional data base:

1. The extensional processor can be realized by a conventional data base management system.
2. Because the theorem prover never accesses the extensional data base, the intensional data base can be compiled using the theorem prover as a once-only

compiler. This means that at query evaluation time there is no need for a theorem prover, nor are there the usual problems involving search which plague current theorem proving systems.

This paper is essentially a survey of some of the results in [Reiter 1977a]. As such, it is necessarily impressionistic, so that no proofs are given for the theorems stated, and the basic approach to query evaluation which decouples the theorem prover from the extensional processor is described by means of an extended example. A rigorous presentation is contained in [Reiter 1977a] to which the interested reader is referred for the painful details.

2. DATA BASES

The results of this paper apply only to first order data bases with the following properties:

(i) The data base consists of finitely many twffs (typed well formed formulae). For example, in an inventory domain, such a twff might be
(x/Manufacturer)(y/Part)(z/Part)Manufactures x,y ∧ Subpart z,y ⊃ Supplies x,z
i.e. every manufacturer of a part supplies all its subparts. The restricted universal quantifier (y/Part) may be read "For every y which is a Part". The restrictions Manufacturer and Part are called types, and are distinguished monadic predicates. If τ is such a type, then $(x/\tau)W$ is an abbreviation for $(x)\tau x \supset W$. We shall later require the notion of a restricted existential quantifier (Ex/τ) which may be read "there is an x in τ". $(Ex/\tau)W$ is an abbreviation for $(Ex)\tau x \wedge W$. We denote by $|\tau|$ the set of all constants which satisfy the type τ. Thus, |Part| might be {gadget-1,widget-3, bolt-49,...,}. In general, a twff has the form $(x_1/\tau_1)\ldots(x_n/\tau_n)W$ for $n \geq 0$ where W is any quantifier-free ordinary first order formula containing no function signs,

and $\tau_1,\ldots,\tau_n$ are types. Notice that no existentional quantifiers are permitted - all twffs are universally quantified. In the case that the twff has no quantifiers, it is an ordinary ground first order formula.

(ii) There are only finitely many constant signs. Constant signs denote individuals of the data base e.g. bolt-49, Acme-manufacturers, etc.

(iii) Equality is a distinguished predicate. We assume that the data base contains the following equality axioms:

E1. $(x)x=x$

E2. $(x)(y)\ x=y \supset y=x$

E3. $(x)(y)(z)\ x=y \wedge y=z \supset x=z$

E4. For each n-ary predicate sign P

$$(x_1)\ldots(x_n)(x_1')\ldots(x_n')\ x_1=x_1' \wedge\ldots\wedge x_n=x_n' \wedge Px_1,\ldots,x_n \supset Px_1',\ldots,x_n'$$

In addition, if $c_1,\ldots,c_p$ are all of the constant signs of the data base, then the following domain closure axiom applies:

DC. $(x)[x=c_1 \vee x=c_2 \vee\ldots\vee x=c_p]$.

The domain closure axiom restricts the universe of discourse to just those individuals denoted by the constant signs of the theory. In the intended interpretation, answers to queries will be formulated exclusively in terms of these finitely many individuals.

Finally, we assume that for each constant sign c, the ground equality literal c=c is in the data base, and for each pair of distinct constant signs c,c' the inequality literal $c \neq c'$ is in the data base. Intuitively, as far as the data base is concerned, two constant signs are treated as equal iff they are identical syntactic objects.

Let DB be a data base as defined above, and let EDB be the set of ground literals of DB. EDB will be called the extensional data base. The intensional data base is defined to be IDB = DB - EDB. Intuitively, the EDB is a set of specific facts like "John Doe teaches Calculus 103", while the IDB is a set of general facts like "All widgets are manufactured by Foobar Inc." or "John Doe teaches Calculus 102 or Bill Jones teaches Calculus 103 (but I don't know which)" together with the equality and domain closure axioms.

3. QUERIES AND ANSWERS

A query is any expression of the form

$$\langle x_1/\tau_1,\ldots,x_n/\tau_n \mid (q_1y_1/\theta_1)\ldots(q_my_m/\theta_m)W(x_1,\ldots,x_n,y_1,\ldots,y_m)\rangle \qquad (1)$$

where (q_iy_i/θ_i) is (y_i/θ_i) or (Ey_i/θ_i), the τ's and θ's are types, and $W(x_1,\ldots,x_n,y_1,\ldots,y_m)$ is a quantifier-free formula containing no function signs and whose variables are $x_1,\ldots x_n,y_1,\ldots,y_m$. For brevity, we shall usually denote the typical query (1) by $\langle \vec{x}/\vec{\tau} \mid (q\vec{y}/\vec{\theta})W(\vec{x},\vec{y})\rangle$.

Intuitively, (1) denotes the set of all n-tuples $\vec{x}$ such that $\vec{x} \in |\vec{\tau}|$[1] and such that $(q\vec{y}/\vec{\theta})W(\vec{x},\vec{y})$ is true.

As an example, consider an inventory domain and the request "Give those manufacturers who supply all widgets". This might be represented in our query language by

<x/Manufacturer|(y/Widget)Supplies x,y>

Formally, let DB be a data base as defined in Section 2, and

[1] If $\vec{\tau} = \tau_1,\ldots,\tau_n$ is a sequence of types, then $|\vec{\tau}| = |\tau_1| \times\ldots\times |\tau_n|$

$Q = \langle \vec{x}/\vec{\tau} | (q\vec{y}/\vec{\theta})W(\vec{x},\vec{y}) \rangle$ a query. A set of n-tuples $\{\vec{c}^{(1)},\ldots,\vec{c}^{(r)}\}$ is an <u>answer</u> to Q (with respect to DB) iff

1. $\vec{c}^{(i)} \in |\vec{\tau}|$ $i=1,\ldots,r$ and
2. $DB \vdash \bigvee_{i \leq r} (q\vec{y}/\vec{\theta})W(\vec{c}^{(i)},\vec{y})$

Notice that if $\{\vec{c}^{(1)},\ldots,\vec{c}^{(r)}\}$ is an answer to Q and $\vec{c}$ is any n-tuple of constants such that $\vec{c} \in |\vec{\tau}|$ then so also is $\{\vec{c}^{(1)},\ldots,\vec{c}^{(r)},\vec{c}\}$ an answer to Q. This suggests the need for the following definitions:

A is a <u>minimal</u> answer to Q iff no proper subset of A is an answer to Q. If A is minimal, then if $|A| = 1$, A is a <u>definite</u> answer to Q. Otherwise A is an <u>indefinite</u> answer to Q. Instead of denoting an indefinite answer by $\{\vec{c}^{(1)},\ldots,\vec{c}^{(r)}\}$, we prefer the more suggestive notation $\vec{c}^{(1)} +\ldots+ \vec{c}^{(r)}$. Indefinite answers have interpretation:

$\vec{x} = \vec{c}^{(1)}$ or $\vec{x} = \vec{c}^{(2)}$ or...or $\vec{x} = \vec{c}^{(r)}$ but there is not enough information, given DB, to determine which. We shall sometimes refer to expressions of the form $\vec{c}^{(1)} +\ldots+ \vec{c}^{(r)}$ as <u>disjunctive tuples</u>. Finally,we denote by $\|Q\|$ the set of minimal answers to Q.

In order to better understand why the possibility of indefinite answers must be entertained, consider the following fragment of a kinship data base:

<u>Example 3.1</u>

<u>IDB</u>

(x/Male)(y/Human)Brother x,y $\supset$ Sibling x,y

(xz/Male)(y/Human)(w/Female)Uncle x,y $\wedge$ Father z,y $\wedge$ Mother w,y

$\supset$ Brother x,z $\vee$ Brother x,w

<u>EDB</u>

Uncle a,b Father c,b Mother d,b Brother a,e

Consider the query "Who are all of a's siblings?"

Q = <x/Human|Sibling a,x>

Then ||Q|| = {e, c+d}

i.e. e is a sibling of a. Moreover either c is a sibling of a or d is, but there is not enough information available to determine which is the case.

4. EXISTENTIAL QUERIES AND EQUALITY

Recall that a data base was defined, in part, to contain the equality axioms E1 - E4, and the domain closure axiom DC. The presence of these axioms will clearly prove disastrous for any theorem proving approach to query evaluation. The use of proof procedures with "built in" equality e.g. paramodulation [Robinson and Wos 1969] will be of little value since the domain closure axiom will still be present and for data bases with a large number of constants, this axiom will inevitably lead to unfeasible computations. Fortunately, for existential queries i.e. queries of the form $\langle \vec{x}/\vec{\tau} | (E\vec{y}/\vec{\theta})W(\vec{x},\vec{y})\rangle$, these axioms turn out to be irrelevant.

Theorem 4.1

Let E(DB) be the equality and domain closure axioms of a data base DB, and let Q be an existential query. Then A is an answer to Q with respect to DB iff it is an answer to Q with respect to DB - E(DB) i.e. the equality and domain closure axioms are irrelevant to existential query evaluation.

5. REDUCTION OF ARBITRARY QUERIES TO EXISTENTIAL QUERIES

As we saw in the previous section, equality poses no difficulties for existential queries. Unfortunately, Theorem 4.1 fails for arbitrary queries. To see why, consider the following data base:

Equality axioms E1 - E4.

DC $(x)x=a'$

Pa,a

$|\tau| = \{a\}$

i.e. a data base with a single constant a and a single type τ. Then $\|\langle x/\tau|(y/\tau)Px,y\rangle\| = \{a\}$. But $(y/\tau)Pa,y$ i.e. $(y)\tau y \supset Pa,y$ is not provable without the domain closure axiom DC.

The approach which we adopt is to reduce arbitrary queries to existential ones by invoking the "projection" and "division" operators which we now define. Let $Q = \langle \vec{x}/\vec{\tau},z/\psi|(q\vec{y}/\vec{\theta})W(\vec{x},\vec{y},z)\rangle$. The quotient of $\|Q\|$ by z, $\Delta_z\|Q\|$, is a set of disjunctive tuples and is defined as follows:

$\vec{c}^{(1)} +\ldots+ \vec{c}^{(m)} \in \Delta_z\|Q\|$ iff

1. For all $\vec{a} \in |\psi|^m$, $(\vec{c}^{(1)},a_1)+\ldots+(\vec{c}^{(m)},a_m)$ is an answer (not necessarily minimal) to Q (and hence some sub-disjunctive tuple of $(\vec{c}^{(1)},a_1) +\ldots+ (\vec{c}^{(m)},a_m)$ is an element of $\|Q\|$), and
2. For no i, $1\le i\le m$, does $\vec{c}^{(1)} +\ldots+ \vec{c}^{(i-1)} + \vec{c}^{(i+1)} +\ldots+ \vec{c}^{(m)}$ have property 1.

(There is a slight abuse of notation here. If $\vec{c} = (c_1,\ldots,c_n)$, then $(\vec{c},a)$ is intended to denote $(c_1,\ldots,c_n,a)$.) The operator Δ_z is called the division operator with respect to z and is an appropriate generalization of the division operator of [Codd 1972].

The projection of Q with respect to z, $\pi_z\|Q\|$, is a set of disjunctive tuples and is defined as follows:

$\vec{c}^{(1)} +\ldots+ \vec{c}^{(m)} \in \pi_z\|Q\|$ iff

1. There exist constant signs $a_j^{(i)} \in |\psi|, j=1,\ldots,r_i$ $i=1,\ldots,m$ such that $\bigsqcup_{j\le r_i}\bigsqcup_{i\le m}(\vec{c}^{(i)},a_j^{(i)}) \in \|Q\|$ and
2. For no i, $1\le i\le m$, does $\vec{c}^{(1)} +\ldots+ \vec{c}^{(i-1)} + \vec{c}^{(i+1)} +\ldots+ \vec{c}^{(m)}$ have property 1.

The operator π_z is called the projection operator with respect to z and is an appropriate generalization of the projection operator of [Codd 1972].

The following theorem indicates the importance of these operators:

Theorem 5.1

If $W(\vec{x},y)$ is a (not necessarily quantifier-free) formula with free variables $\vec{x}$ and y, then

1. $\|<\vec{x}/\tau\,|\,(y/\theta)W(\vec{x},y)>\| = \Delta_y\|<\vec{x}/\vec{\tau},y/\theta\,|\,W(\vec{x},y)>\|$
2. $\|<\vec{x}/\tau\,|\,(Ey/\theta)W(\vec{x},y)>\| = \pi_y\|<\vec{x}/\vec{\tau},y/\theta\,|\,W(\vec{x},y)>\|$

Using Theorem 5.1 we can now represent an arbitrary query as the appropriate application of projection and division operators to an existential query. For example,

$\|<x/\tau_1\,|\,(Ey/\tau_2)(z/\tau_3)(Ew/\tau_4)W(x,y,z,w)>\| = \pi_y\Delta_z\|<x/\tau_1,y/\tau_2,z/\tau_3\,|\,(Ew/\tau_4)W(x,y,z,w)>\|$

In view of Theorem 5.1 it is sufficient to devise techniques for evaluating existential queries only, in which case, by Theorem 4.1, we can eliminate the equality and domain closure axioms from the data base.

6. ANSWERING EXISTENTIAL QUERIES

The approach of this paper is designed for very large data bases in which the vast majority of facts are extensional i.e. |EDB|>>|IDB| with |EDB| very large. Under these circumstances, conventional theorem proving approaches (e.g. [Minker et al. 1972]) are likely to be quite inefficient since the theorem prover is intermingling access to both the IDB and EDB. As an alternative, Figure 1 illustrates our proposed system design. There are several points worth noting:

1. As its name implies, the extensional query evaluator evaluates queries in our query language, but only with respect to the EDB. As such, it need not be a conventional theorem prover.

2. The most significant observation is that the EDB and IDB processors are completely decoupled. The IDB, but not the EDB, is invoked during the theorem proving process. Conventional theorem provers are notoriously inefficient. Since, in applications to large data bases, we can expect |EDB|>>|IDB|, the last thing we want is to require of the theorem prover that it have to look at the EDB. Moreover, there are far more efficient non theorem proving techniques for extensional query evaluation, e.g. relational query evaluation [Palermo 1974, Reiter 1976]. In effect, this decoupling of the EDB and IDB processors relegates the search task over the IDB to the theorem prover, and the "search-free computational" task over the EDB to the extensional query evaluator.
3. The result of the theorem proving process is a proof search tree from which a set of queries $Q_1,\ldots,Q_m$ can be extracted. These are extensionally evaluated and the results unioned to obtain the answers to the original query Q.

We cannot, in the limited space of this paper, formally describe and justify the approach to deductive question-answering of Figure 1. Instead, we shall try to convey its basic flavour by means of an example. The interested reader is referred to [Reiter 1977a] for particulars.

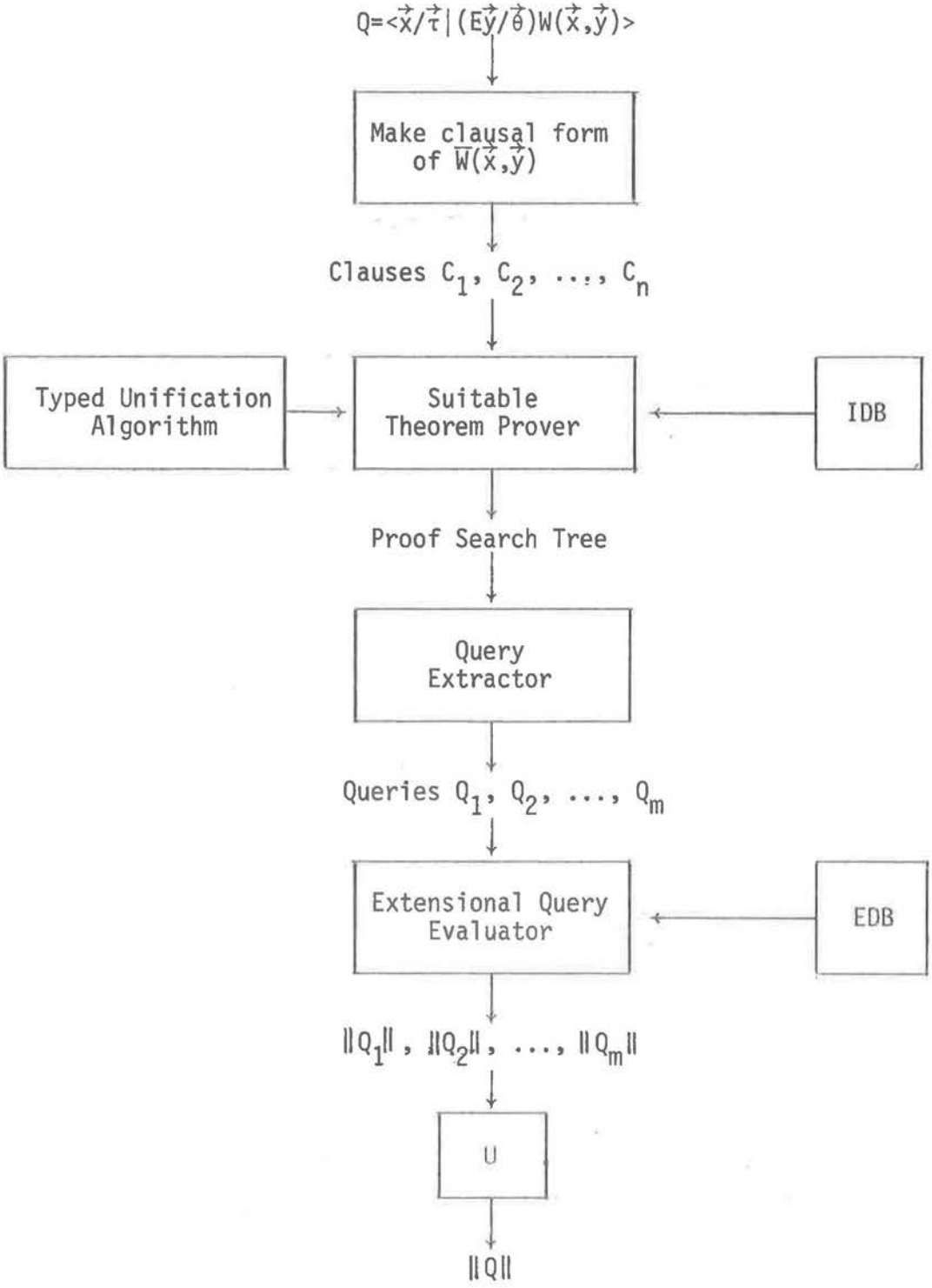

Figure 1

System Overview

Example 6.1

We consider a simple fragment of an education domain.

IDB

(1) A teaches all calculus courses.

(z/Calculus)Teach A,z

(2) B teaches all computer science courses

(y/CS)Teach B,y

(3) If teacher u teaches course v and student w is enrolled in v,

then u is a teacher of w.

(u/Teacher)(v/Course)(w/Student)Enrolled w,v ∧ Teach u,v ⊃ Teacher-of w,u

EDB

Teach	x	y
	A	P100
	B	P200
	C	P300
	D	H100
	D	H200

Enrolled	x	y
	a	C100
	a	P300
	a	CS100
	b	C200
	b	CS200
	b	CS300
	c	H100
	c	C100
	d	H200
	d	P200
	d	P300

|Teacher| = {A,B,C,D}

|Student| = {a,b,c,d}

|Course| = {C100,C200,CS100,CS200,CS300,H100,H200,P100,P200,P300}

|Calculus| = {C100,C200}

|CS| = {CS100,CS200,CS300}

Consider the query "Who are a's teachers?"

Q = <x/Teacher|Teacher-of a,x>

We start by treating (Ex/Teacher)Teacher-of a,x as a theorem to be proved with DB as hypotheses. Using the usual refutation approach, we create the clausal form of its negation i.e. $\overline{\text{Teacher-of}}$ a,x and associate, with the variable x of this clause, the type Teacher. Now consider attempting a linear refutation using $\overline{\text{Teacher-of}}$ a,x as top clause. There are two possibilities:

(i) This top clause could be resolved against a unit of the EDB, or

(ii) It could be resolved against a clause (in this case the clausal form[1] of (3)) of the IDB.

Our approach is to admit both possibilities, as in Figure 2, but to perform just the second. Literals enclosed in curly brackets represent literals which possibly might have been, but were not, resolved away against the EDB. The label on the right branch of Figure 2 indicates that clause of the IDB against which $\overline{\text{Teacher-of}}$ a,x was resolved. The left branch is unlabeled, indicating that we could have tried to resolve $\overline{\text{Teacher-of}}$ a,x against the EDB, but we have postponed this attempt by instead placing the literal within curly brackets. This left node is now closed, since there are no remaining literals to resolve against the IDB. The right node, with clause $\overline{\text{Enrolled}}$ a,v $\vee$ $\overline{\text{Teach}}$ x,v remains open. Again there are two possibilities:

(i) Resolve $\overline{\text{Teach}}$ x,v against the EDB, or

(ii) Resolve it against clauses (1) or (2) of the IDB.

Figure 3 represents these possibilities, again postponing the resolution operation against the EDB, as indicated by the unlabeled left branch. The branches labeled (1) and (2) correspond to possibility (ii).

It is clear that we can continue expanding nodes in this way. If

$N = \{E_1,\ldots,E_k\} : L_1 \vee \ldots \vee L_p$

is a typical such node, then N will have successors $N_o, N_1, \ldots, N_r$ where

$N_o = \{E_1,\ldots,E_k,L_p\} : L_1 \vee \ldots \vee L_{p-1}$

---

[1] In converting IDB formulae to clausal form, one eliminates the quantifiers. Since all quantifiers are restricted by types, these types must be associated with their corresponding variables in the clausal form.

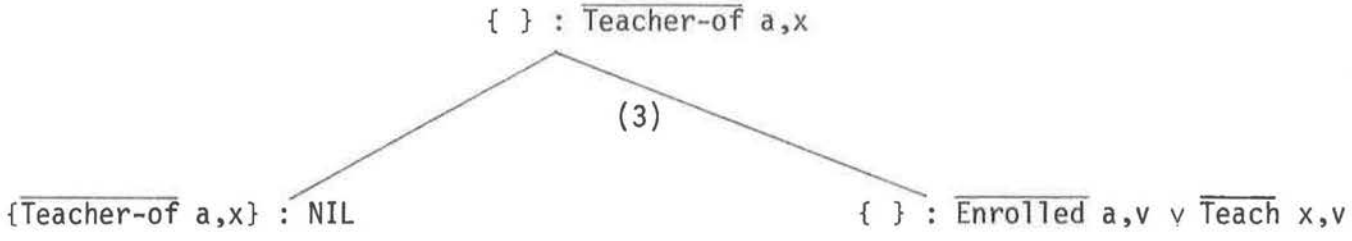

Figure 2

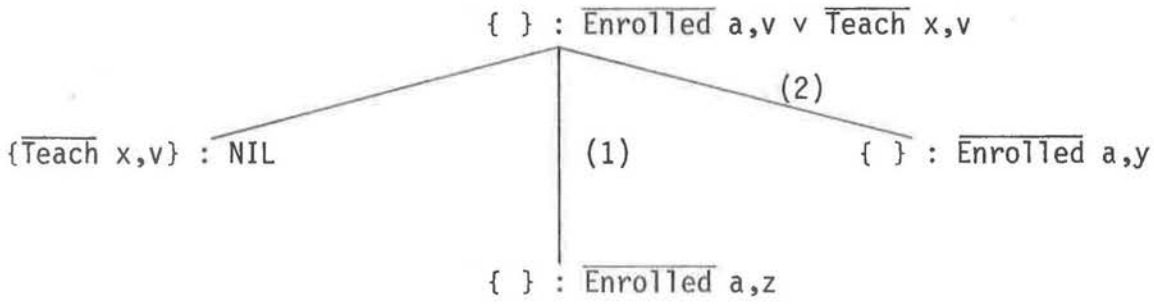

Figure 3

indicating that we are postponing any attempted resolution of L_p against a unit of the EDB. For $i = 1,\ldots,r$

$N_i = \{E_1\sigma_i,\ldots,E_k\sigma_i\} : R_i$

where R_i is the resolvent obtained by resolving the clause $L_1 \vee \ldots \vee L_p$ upon its rightmost literal L_p against a clause of the IDB and σ_i is the corresponding unifying substitution. Notice that we are here using a clause ordering linear resolution strategy in which only rightmost literals in a deduction are resolved upon. When suitably formalized, such a strategy can be shown to be complete [Reiter 1971].

For the example at hand, we can continue expanding nodes until eventually no further expansion is possible. Figure 4 shows the resulting fully expanded tree. The only new feature in this figure is the introduction of an answer literal ANS x [Green 1969] whose function is to record the substitutions being made for the variable x. Clearly there is a need for this book-keeping device since any substitution made for x is a possible answer to Q.

Now consider a typical terminal node in Figure 4, say {ANS x, $\overline{\text{Enrolled}}$ a,v, $\overline{\text{Teach}}$ x,v} : NIL. This means that the non answer literals have yet to be resolved away against the EDB. In other words, the query

Q_1 = <x/Teacher|(Ev/Course)Enrolled a,v $\wedge$ Teach x,v>

when extensionally evaluated yields a set of answers to the original query Q. Similarly, the remaining terminal nodes yield the following queries for extensional evaluation:

Q_2 = <x/Teacher|Teacher-of a,x>

Q_3 = <x/Teacher|(Ez/Calculus)Enrolled a,z $\wedge$ x=A>

Q_4 = <x/Teacher|(Ey/CS)Enrolled a,y $\wedge$ x=B>

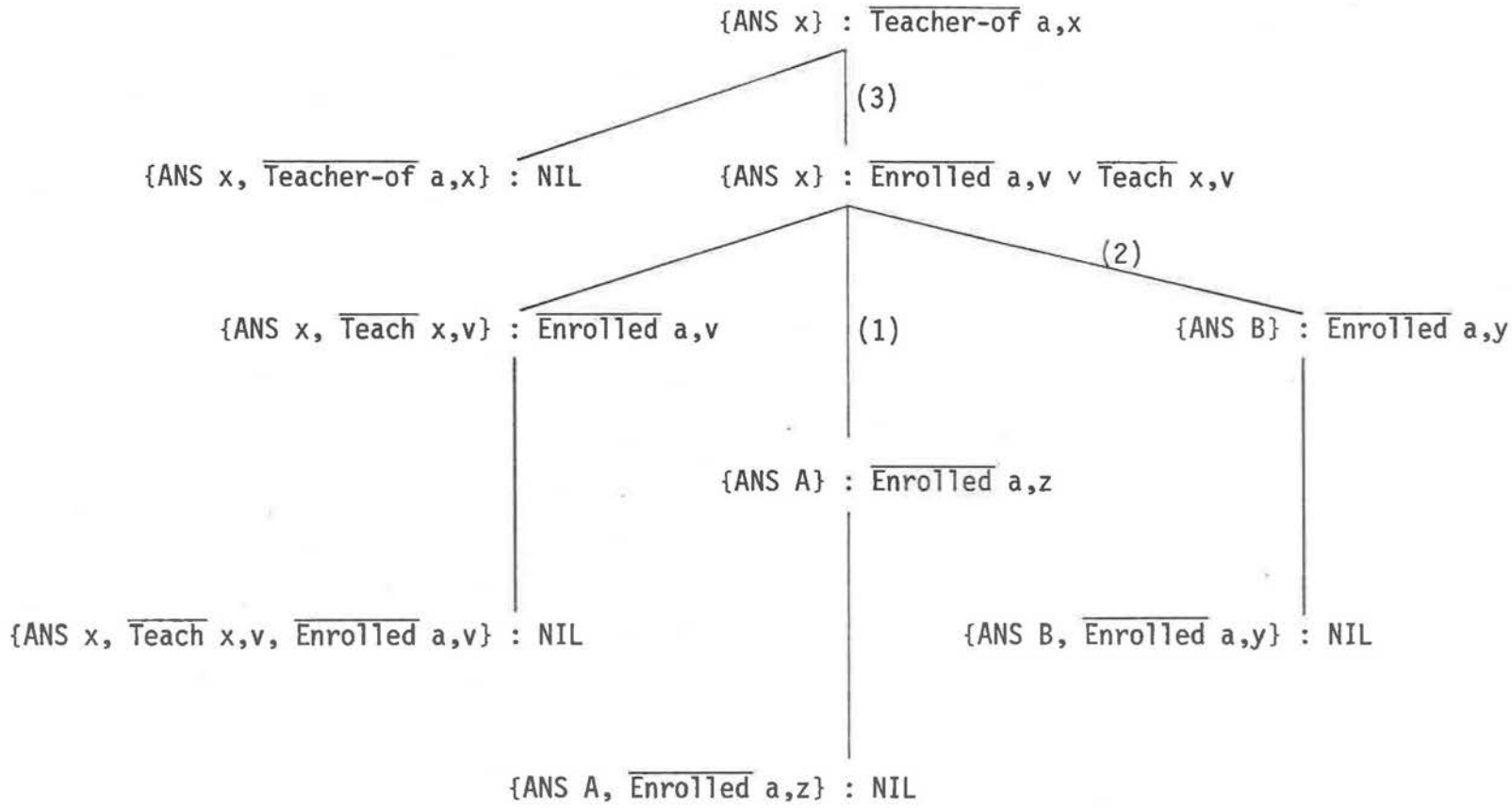
{ANS x} : ¬Teacher-of a,x
(3)
{ANS x, ¬Teacher-of a,x} : NIL
{ANS x} : ¬Enrolled a,v ∨ ¬Teach x,v
(2)
{ANS x, ¬Teach x,v} : ¬Enrolled a,v
(1)
{ANS B} : ¬Enrolled a,y
{ANS A} : ¬Enrolled a,z
{ANS x, ¬Teach x,v, ¬Enrolled a,v} : NIL
{ANS B, ¬Enrolled a,y} : NIL
{ANS A, ¬Enrolled a,z} : NIL

Figure 4

These queries have respective extensional values {C}, ϕ, {A}, {B} the union of which yields

$\|Q\| = \{A,B,C\}$

Notice how the intensional and extensional processors are totally decoupled under this approach. Figure 4 represents the first phase of query evaluation. This is the only task allocated to the theorem prover and nowhere in this process does the theorem prover probe the EDB. The second phase requires the extensional evaluation of the queries Q_1, Q_2, Q_3 and Q_4. This can be done by a relational data base management system and is certain to be far more efficient for large EDBs than any theorem proving technique. For an approach to extensional query evaluation designed for the query language of this paper and which optimizes for equality see [Reiter 1976].

7. COMPLETENESS OF THE QUERY EVALUATION PROCESS

As this paper is necessarily impressionistic, we have been deliberately vague in Section 6 about the nature of the theorem prover that is required. Moreover, there are a number of features which the simple example of that section fails to illustrate:

1. The derivation of indefinite answers.
2. How the types associated with the variables of a clause affect the unification algorithm.
3. The treatment of multiple clauses arising from a query.

In [Reiter 1977a] all of these issues are made precise. Once this has been done, it is possible to prove the following completeness result:

Provided the extensional query evaluator returns all and only the answers to a given query and provided an appropriate theorem prover is used for the intensional processing, then the approach of this paper is complete i.e. all and only the answers to a given query will be returned, including indefinite answers should they arise.

In a very real sense, this completeness result must be taken with a grain of salt, for in order to properly make use of it the proof search tree generated by the theorem prover must be finite, as indeed that of Figure 4 is. In our view a data base must be so structured so as to guarantee the finiteness of all such trees i.e. there is a design issue here. For some suggestions as to how to appropriately structure a data base in order to guarantee finite computations at query evaluation time, see [Reiter 1977a].

8. COMPILING THE IDB

We have shown that query evaluation may be decomposed into an intensional processor involving a theorem prover which computes only on the IDB, and an extensional processor computing only on the EDB. One very nice feature of this decomposition is that it is now possible to compile the IDB using the theorem prover as a once-only compiler.

The basic idea is quite simple. For each predicate of the data base, say the predicate Teach of the example, determine all "proofs" with top clause Teach x,y as well as with top clause $\overline{\text{Teach}}$ x,y. Notice that these "proofs" involve

only clauses of the IDB. Next store all such trees, for all predicates, on an external file, and discard both the IDB and the theorem prover. Then at query evaluation time, read in all of the "proofs" for those signed predicates occurring in the query. These trees can then be appropriately combined to yield all of the "proofs" required by the query.

It turns out that this compilation process together with the resulting query evaluation are considerably simplified under the so-called closed world assumption (CWA). In order to illustrate what is involved we shall assume that the reader is familiar with the material and notation in [Reiter 1977b]. In particular, we shall exploit the fact that the set of CWA answers to an arbitrary query can be computed by applying the set operations of union, intersection and difference and the relational algebra operation of projection to the open world assumption (OWA) answers to atomic queries. Thus, CWA query evaluation reduces to OWA evaluation for atomic queries. Now an atomic query has the form

$Q = \langle \vec{x}/\vec{\tau} \mid (E\vec{y}/\vec{\theta})Pt_1,\ldots,t_n \rangle$

where each t is an x, a y, or a constant, and P is a predicate sign. Suppose that we had available all "proofs" of the literal $Pu_1,\ldots,u_n$ using only the clauses of the IDB, where each variable u_i has type $\mathbb{U}$, the universal type i.e. $|\mathbb{U}|$ = the set of all constants. Then to compute $\|Q\|_{OWA}$ simply substitute t_i for u_i in these proofs ensuring that the type of t_i is consistent with the type of u_i. Then, as in Section 6, form appropriate queries using the terminal nodes in the resulting proof tree, and extensionally evaluate these queries. The method is best explained by an example.

Example 8.1

We shall treat the data base of Example 6.1 in closed world mode. To begin, we compile the predicate Teach i.e. we determine all "proofs" of Teach u_1,u_2 i.e. all

"refutations" with top clause $\overline{\text{Teach}}\ u_1,u_2$. Here u_1 and u_2 both have type U. The "refutations" use only the clauses of IDB just as we did in Section 6. Figure 5 shows the resulting fully expanded search tree. The types associated with each free variable at a given node are indicated. Now suppose we wish to determine $\|Q\|_{OWA}$ where

Q = <x/Teacher|Teach x, CS100>

To do so, substitute x of type Teacher for u_1 and substitute CS100 for u_2 in Figure 5. The substitution of x for u_1 satisfies the type restrictions on u_1 throughout the tree. The substitution of CS100 for u_2 violates the type restriction on u_2 at node 2 since CS100 $\notin$ |Calculus|, so node 2 cannot contribute to the evaluation of Q. Nodes 1 and 3 do contribute, so we form the two queries:

Q_1 = <x/Teacher|Teach x,CS100>

Q_3 = <x/Teacher|x = B>

Q_1 and Q_3 extensionally evaluate to ϕ and {B} respectively, whence $\|Q\|_{OWA}$ = {B}.

It should be clear that we had no use for the entire tree of Figure 5 - only the terminal nodes were necessary. Moreover, the relevant information contained in these nodes is more succinctly representable by the following three formulae with free variables u_1 and u_2

(i) Teach u_1,u_2 $\quad u_1/U\ \ u_2/U$

(ii) $u_1 = A \wedge u_2 = u_2$ $\quad u_1/U\ \ u_2/\text{Calculus}$

iii) $u_1 = B \wedge u_2 = u_2$ $\quad u_1/U\ \ u_2/CS$

We shall refer to these three formulae as the compiled form of the predicate Teach. As a further example consider

Q = <x/Teacher|(Ey/CS)Teach x,y>

Substitute x of type Teacher for u_1 and y of type CS for u_2 in (i), (ii) and (iii) above. This yields a type inconsistency in (ii), so we obtain two queries:

Q_1 = <x/Teacher|(Ey/CS)Teach x,y>

Q_3 = <x/Teacher|(Ey/CS)x = B $\wedge$ y = y>

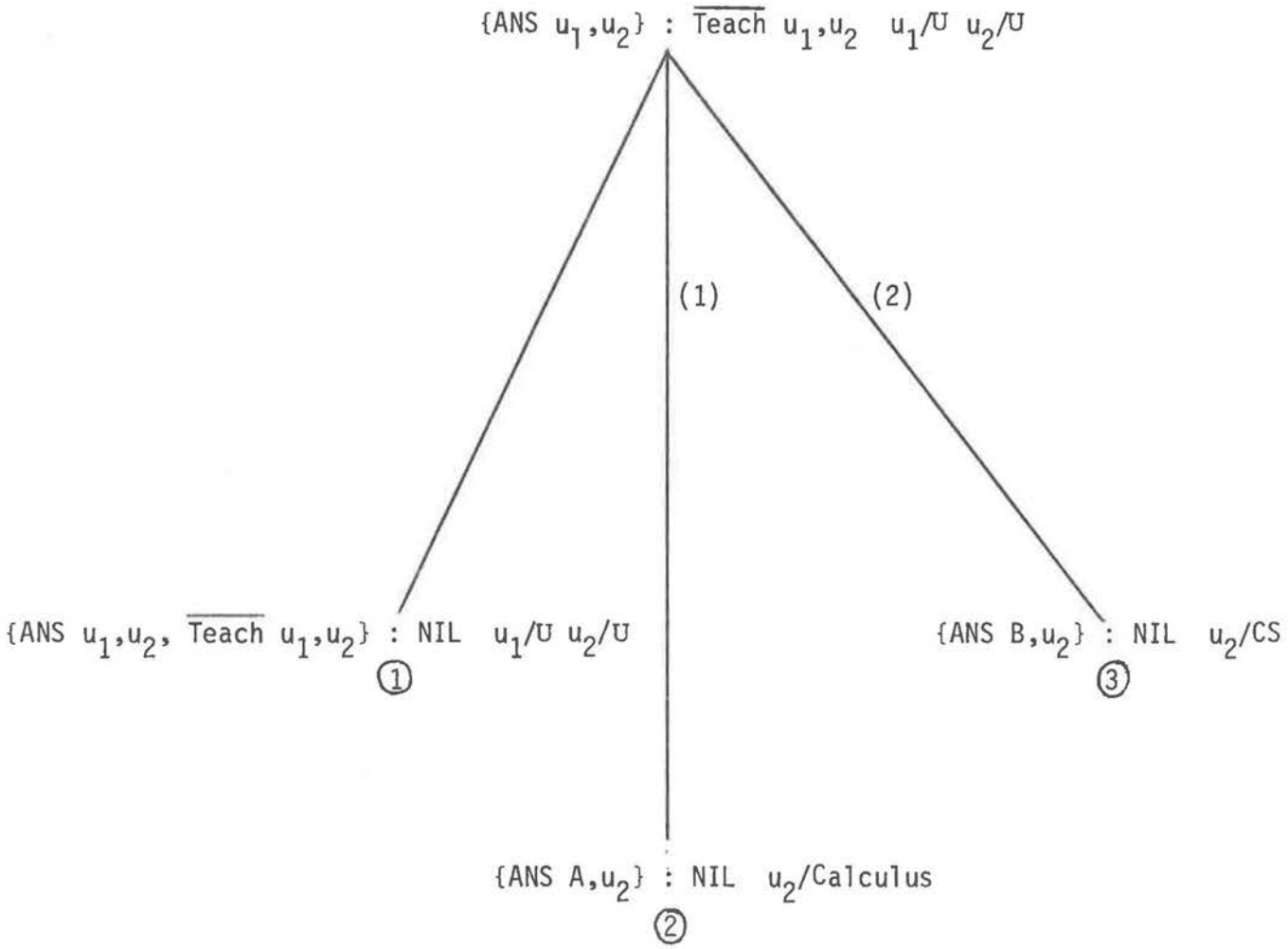

Figure 5

The refutation search tree for Teach

These extensionally evaluate to ϕ and {B} respectively, whence $||Q||_{OWA}$ = {B}.

Next we compile the predicate Teacher-of. Figure 6 contains the refutation search tree. From its terminal nodes we obtain the following compiled form for Teacher-of:

Teacher-of u_1,u_2 u_1/U u_2/U

(Ev/Course)Teach u_2,v $\wedge$ Enrolled u_1,v u_1/Student u_2/Teacher

(Ev/Calculus)Enrolled u_1,v $\wedge$ u_2 = A u_1/Student

(Ev/CS)Enrolled u_1,v $\wedge$ u_2 = B u_1/Student

consider evaluating

Q = <x/Student|Teacher-of x,B>

Substituting x for u_1 and B for u_2 in the compiled form of Teacher-of yields the following queries for extensional evaluation:

Q_1 = <x/Student|Teacher-of x,B>

Q_2 = <x/Student|(Ev/Course)Teach B,v $\wedge$ Enrolled x,v>

Q_3 = <x/Student|(Ev/Calculus)Enrolled x,v $\wedge$ B = A>

Q_4 = <x/Student|(Ev/CS)Enrolled x,v $\wedge$ B = B>

These extensionally evaluate to ϕ, {d}, ϕ and {a,b} whence $||Q||_{OWA}$ = {a,b,d}.

Finally, consider evaluating the following non atomic query under the CWA:

Q = <x/Teacher|Teacher-of a,x $\wedge$ $\overline{\text{Teach}}$ x,CS100>

Then

$||Q||_{CWA} = ||Q_1||_{OWA} \cap (|\text{Teacher}| - ||Q_2||_{OWA})$

where

Q_1 = <x/Teacher|Teacher-of a,x>

Q_2 = <x/Teacher|Teach x,CS100>

Evaluating Q_1 using the compiled form of Teacher-of yields {A,B,C}. We have already evaluated Q_2 at the beginning of this example, yielding {B}. Hence $||Q||_{CWA}$ = {A,C}.

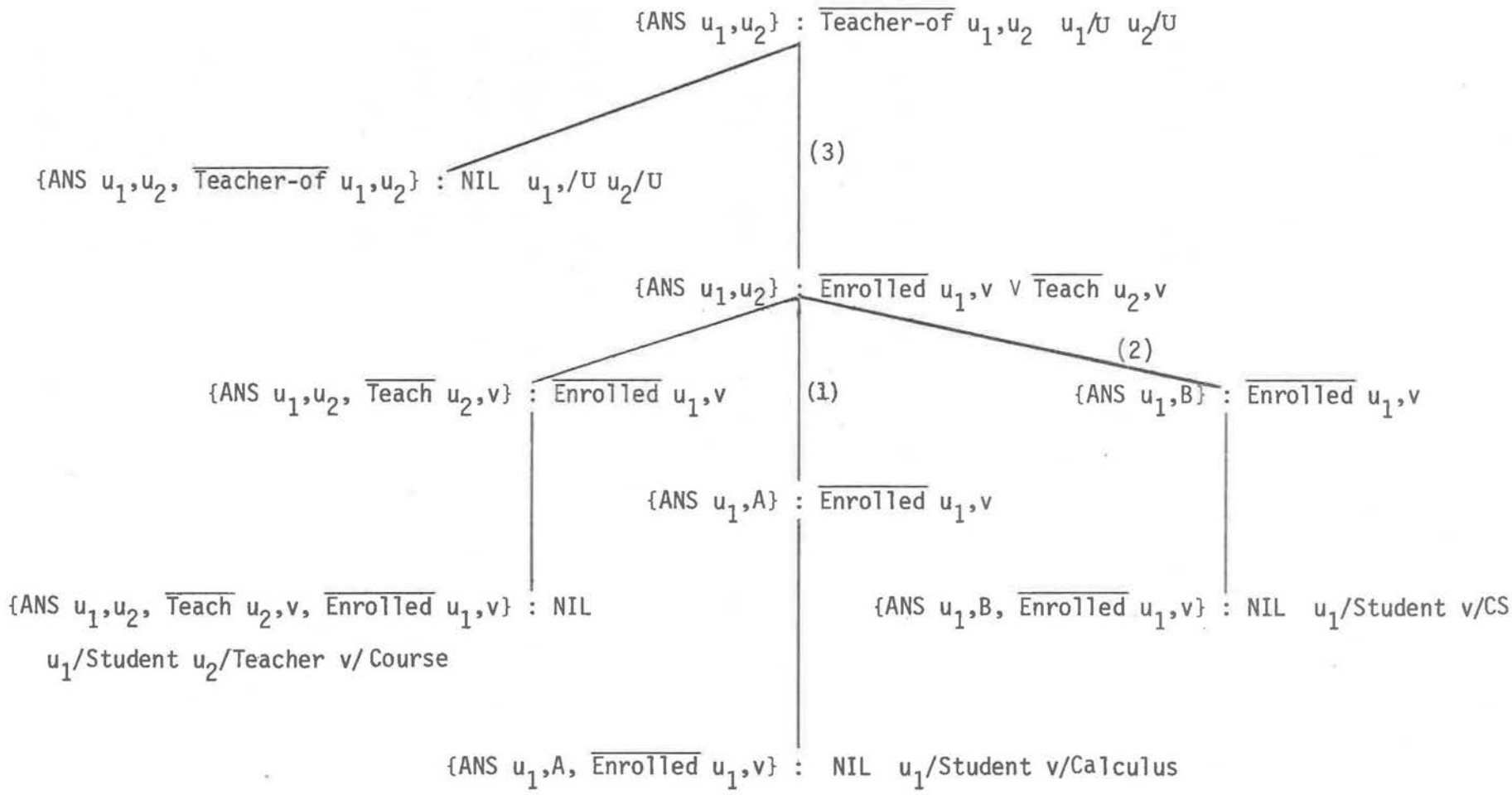

Figure 6

The refutation search tree for Teacher-of

There are a number of advantages to this approach of compiling the IDB:

(i) The time required for query evaluation is reduced since there is no need to search for all possible proofs.

(ii) Compilation completely eliminates the need for a theorem prover at query evaluation time. Given the unpredictable nature of current theorem provers we can expect more stable performance of a deductive question-answering system that makes no use of one.

(iii) The compilation process can be effected by a suitably designed interactive theorem prover. This can provide for a far greater measure of control over the deductive mechanism than is currently possible under autonomous theorem proving systems. In particular, the data base designer will be in a position to interactively exploit his or her knowledge of the semantics of the domain to prune fruitless or infinite deduction paths, to apply optimizing transformations, and to recognize redundant or duplicate deductions. Moreover, the design of such an interactive system is far simpler than that of an autonomous theorem prover and requires significantly less code. Finally, since efficiency considerations for such an interactive theorem prover are irrelevant given that it is functioning as a once-only compiler, it's implementation is even further simplified. And of course, once the compilation is completed, the compiler may be expunged from the system.

(iv) The query language of this paper is set oriented i.e. we seek all answers to a given query. Moreover the techniques for query evaluation which we have proposed are specifically directed at computing sets of answers. One might conclude from this that the evaluation of "single answer" queries will require a substantially different approach. By a "single answer" query we mean one whose appropriate answer is "yes", "no" or "I don't know". (This latter cannot arise under the CWA.) For example, "Is A a teacher of b?" i.e. Teacher-of b,A or "Does A teach calculus?" i.e. (Ex/Calculus)Teach A,x. It should be clear, however,

that if the IDB is compiled, it is trivial to evaluate "single answer" queries. Merely instantiate in turn each formula of the appropriate compiled forms of the predicates, and send that formula off to the extensional evaluator to extensionally test its truth value. For example, to evaluate Teacher-of b,A we retrieve the compiled form of Teacher-of and extensionally test each of the following formulae in turn, returning "yes" if and when one of them tests true:

Teacher-of b,A

(Ev/Course)Teach A,v $\wedge$ Enrolled b,v

(Ev/Calculus)Enrolled b,v $\wedge$ A = A

(Ev/CS)Enrolled b,v $\wedge$ B = A

The evaluation of "single answer" queries in the case of "compound" queries is slightly more complicated. By a compound query we mean one with more than one literal. For example, "Is anyone a teacher of both a and b?" i.e.

(Ex/Teacher)Teacher-of a,x $\wedge$ Teacher-of b,x.

In this case one must form all possible conjunctions of pairs of formulae in the compiled form to Teacher-of, and extensionally test each of these in turn. Thus we must test

(Ex/Teacher)Teacher-of a,x $\wedge$ Teacher-of b,x

(Ex/Teacher)Teacher-of a,x $\wedge$ (Ev/Course)Teach x,v $\wedge$ Enrolled b,v

(Ex/Teacher)Teacher-of a,x $\wedge$ (Ev/Course)Enrolled b,v $\wedge$ x = A

etc.

One last point. Notice that the concept of a compiler for the IDB is feasible only under an approach to deductive question-answering which completely decouples the IDB and EDB theorem proving processors, as described in Section 6. Any attempt at deductive question-answering by means of a theorem prover which intermingles access to both the IDB and EDB can only run "interpretively" since the set of all possible proofs corresponding to a predicate P will in general be impossibly large in the presence of any sizeable EDB.

ACKNOWLEDGEMENT

This paper was written with the financial support of the National Research Council of Canada under grant A7642.

REFERENCES

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

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

Minker, J., Fishman, D.H., and McSkimin, J.R. (1973). "The Q* Algorithm - A Search Strategy for a Deductive Question-Answering System," Artificial Intelligence, 4, Winter 1973, 225-243.

Palermo, F.P. (1974). "A Data Base Search Problem," in Information Systems, J.T. Tou (Ed.), Plenum Press, New York, 1974, 67-101.

Reiter, R. (1971). "Two Results on Ordering for Resolution with Merging and Linear Format," J.ACM 18, 4 (October 1971), 630-646.

Reiter, R. (1976). "Query Optimization for Question-Answering Systems," Proc. COLING, Ottawa, Canada, June 28-July 2, 1976.

Reiter, R. (1977a) "An Approach to Deductive Question-Answering," Technical Report, Bolt Beranek and Newman Inc., Cambridge, Mass., Sept. 1977, 161 pp.

Reiter, R., (1977b)."On Closed World Data Bases" Technical Report 77-16, Department of Computer Science, Univ. of British Columbia, Vancouver,B.C. Oct. 1977.

Robinson, G.A., and Wos. L. (1969). "Paramodulation and Theorem Proving in First Order Theories with Equality," in Machine Intelligence, Volume 4, B. Meltzer and D. Michie (Eds.), American Elsevier, New York, 1969, 135-150.