A UNIFORM LOGICAL TREATMENT OF QUERIES AND UPDATES[1]

by

T.S.E. Maibaum[2], C.S. dos Santos[3]
and
A.L. Furtado[4]

RESEARCH REPORT CS-80-11

University of Waterloo
Computer Science Dept.
Waterloo, Ontario
Canada N2L 3G1

FEBRUARY 1980

[1]Support from the Conselho Nacional de Desenvolvimento Cientifico e Technologico and the Financiadora de Estudos e Projetos of Brasil as well as support from the Canadian International Development Agency and the Natural Sciences and Engineering Research Council of Canada is gratefully acknowledged.

[2]University of Waterloo
Computer Science Dept.
Waterloo, Ontario, Canada.

[3]Universidade Federal do Rio Grande do Sul - Brasil.

[4]Pontifícia Universidade Católica do Rio de Janeiro - Brasil.

1. Introduction

In this paper, a formal approach to data bases providing a uniform treatment of queries and updates is discussed. The formalism is defined as a many sorted first order predicate calculus incorporating a formalized notion of data base state ([MAI]). The entity relationship model is assumed ([CHE]).

In this formalism, data bases are defined through an axiom set specifying:

a. how states can be transformed by the application of update operations;

b. the valid states and state transitions allowed by the adopted integrity constraints.

A user's request for an update contains no direct references to update operations and is expressed as a formula, using predicates defined in the formalism, whose truth-value the user wishes to be (or to become) true. A theorem prover would then synthesize a sequence of update operations ([SAN]) able to move the data base from whatever is its current state to an adequate target state. This sequence is empty if the user's formula is already true in the current state; on the other hand, the system fails to produce such a sequence if the request is incompatible with the axiom set (data base specification).

Conversely, the data base being in some current state, a user may place a request, again expressed as a formula, for querying the data base in the sense of:

a. simply asking if the formula is true or false, or, in additon

b. finding for each existentially quantified variable (other than the state variable) some constant, if any, for which the formula is true.

Updates and queries are uniformly handled as theorems to be proved with respect to the data base specification. The proof process may result in the synthesis of a sequence of operations (updates) or in the analysis of the current state (queries). A state in our formalism is also denoted by a sequence of operations which builds it from an initial empty state.

With the formalism defined in this paper, the behaviour of a data base may be specified at a conceptual schema level ([ANS]), with no regard to implementation details, so that the implementors are not constrained in their search for efficient options. It also provides a convenient interface to non-professional users, by removing the burden of learning how the data base is operated.

2. Mathematical Preliminaries

2.1 Many-sorted logic

We introduce in this section the necessary definitions and results for defining our modelling technique for data bases. The language used is that of many sorted structures and logic - a simple generalisation of normal logic and model theory.

Let S be a set of sorts (names of different kinds of objects). Let S* be the set of strings defined on S with λ the empty string. An S-sorted structure $\mathcal{D}$ is an object defined by <D,B,G,R> where

(i) $D = \{D_s\}_{s \in S}$ is a family of non-empty sets called the carrier or underlying set of $\mathcal{D}$. D_s is the carrier of sort s;

(ii) $B = \{B_s\}_{s \in S}$ is a family of sets of constants. Each $B_s \subseteq D_s$;

(iii) G is a set of operations so that for each $g \in G$,

$g: D_{s_1} \times \ldots \times D_{s_n} \rightarrow D_s$ for some $\langle s_1 \ldots s_n, s \rangle \in S^* \times S$.

$\langle s_1 \ldots s_n, s \rangle$ is called the type of g, s its sort.

(iv) R is a set of relations so that for each $r \in R$,

$r \subseteq D_{s_1} \times \ldots \times D_{s_n}$ for some $s_1 \ldots s_n \in S^*$. The string $s_1 \ldots s_n$ is called the type of r;

A many-sorted language L is a quadruple (S,C,<F,α>,<P,β>) with:

(i) S a non-empty set of sorts;

(ii) $C = \{C_s\}_{s \in S}$ a family of non-empty sets of constant symbols;

(iii) F a (possibly empty) set of function symbols and a map

$\alpha: F \rightarrow S^* \times S$ assigning to each $f \in F$ its type $\alpha(f) = \langle s_1 \ldots s_n, s \rangle \in S^* \times S$;

(iv) P a (possibly empty) set of predicate symbols and a map

β: P $\to$ S* assigning to each r$\in$P its type $\beta(r) = s_1 \ldots s_n \in S^*$.

We will often write L = (S,C,F,P) and leave α and β implicit.

The above are non-logical symbols and we also assume the familiar logical symbols $\neg$, $\to$, $\forall$, ... and for each sort s$\in$S:

(i) an infinite set V_s of variables of sort s with

$V_{s_1} \cap V_{s_2} = \phi$ if $s_1 \neq s_2$;

(ii) the equality predicate symbol $\approx_s$ of type ss.

The definition of terms and formulas of L are similar to the normal (one sorted) definitions except that each term has a sort associated with it and the types of function or predicate symbols and the sorts of their arguments must match. Also we must interpret the use of variables in formulae, as in $\forall v$ with $v \in V_s$, to be restricted so that they may be replaced only by values of the appropriate sort.

A structure $\mathcal{D}_L = (D_L, C_L, F_L, P_L)$ for L is a structure so that D_L is a family of non-empty sets sorted by S, $C_L = \{C_{L,s}\}_{s \in S}$ so that for each $c \in C_s$ there is $c_L \in C_{L,s}$, for each f$\in$F of type $\langle s_1 \ldots s_n, s \rangle$ there is $f_L \in F_L$ such that f_L: $D_{s_1} \times \ldots \times D_{s_n} \to D_s$, and for each r$\in$P of type $s_1 \ldots s_n$ there is $r_L \in P_L$ such that $r_L \subseteq D_{s_1} \times \ldots \times D_{s_n}$.

The definitions of such concepts as satisfaction, truth, etc. follow the usual one-sorted pattern. We can define a deductive calculus similar to the one-sorted case and prove the usual completeness theorem. Also other basic results (such as compactness, the Lowenheim-Skolem theorem, etc.) still hold for the many-sorted case. In fact, one-sorted logic can be generalised straightforwardly and without loss of any results to the many-sorted case.

2.2 Theorem-proving

The many sorted first order predicate calculus formalism presented and used in this paper as a data base specification language has, among others, the advantage of being suitable for the automatization of certain important functions in data base management, namely, query answering and update transaction synthesis ([SAN]), based directly on the data base specification.

In our formalism, a data base is specified as a set s of axioms which defines all the states (configurations) that are valid in the evolution of the data base. In this context, updates and queries can be performed as a theorem-proving process, in which a theorem t (expressing a query or an update) is proved with respect to s.

For example, if the theorem

$$t_1\text{: } (\exists s)\ exs(a,A,s)$$

is proved, then there exists a valid state in which an entity a exists in the entity set A and the system determines such a state, which implements the desired update.

On the other hand, if the theorem

$$t_2\text{: } isr(a,b,R,cs),$$

where cs denotes the current data base state, is proved, then the answer to the query

"Are entities a and b related via relationship R?"

is affirmative.

As we are using an extension of the first order predicate calculus which preserves properties like validity and satisfiability of WFFs, proof procedures are available for our formalism ([NIL]). In particular, the

resolution method ([ROB]) is the most popular proof procedure for the first order predicate calculus, having many implementations available.

In order to apply the resolution method, the axioms and the negation of the theorem must be converted into clause form, i.e. a conjunction of a set of disjunctive clauses, with all existential and universal quantifiers removed. Each one of the disjunctive clauses has the form

$$P_1 \vee P_2 \vee P_3 \vee \ldots \vee P_n,$$

where each P_i is a (possibly negated) predicate symbol with zero or more arguments, which may or may not be instantiated.

In each step of the application of the resolution method (an iterative method), two clauses $(A \vee p)$ and $(B \vee q)$, where q (or one of its instances) is a negation of the predicate symbol p, or vice-versa, are resolved, giving rise to a derived clause $(A \vee B)$ which is inserted in the set of clauses (A and B being the remaining terms of the resolved clauses.)

If the set of clauses is unsatisfiable (the theorem is valid), then the empty clause is eventually derived and the process stops.

Consider, for example, the following set s of clauses

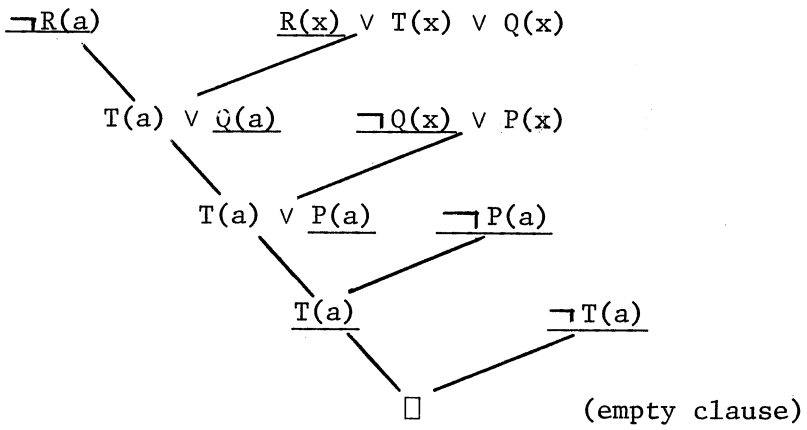
c1) $\neg Q(x) \vee P(x)$

c2) $R(x) \vee T(x) \vee Q(x)$

c3) $\neg P(a)$

c4) $\neg T(a)$

and a theorem t: R(a) to be proved with respect to s. The following is a refutation tree for this example, starting with the negation of the theorem:



So, R(a) is a logical consequence of s.

3. Database Axiomatisation

We present in this section the axiomatisation of a simple data base based on the entity-relationship model ([CHE]). To begin, we specify the many-sorted language used in our example. We will specify the constant, function and relation symbols by adapting the specification method used for specifying abstract data types ([ADJ]). Constants will be presented as in

$$c: \rightarrow s$$

where s is the sort of c. Function symbols will be presented as in

$$f: \ s_1 \times \ldots \times s_n \rightarrow s$$

where $\langle s_1 \ldots s_n, s \rangle$ is the type of f. Relation symbols will be presented as in

$$r: s_1 \times \ldots \times s_n$$

where $s_1 \ldots s_n$ is the type of r.

Sorts: states, entity-names, relation-names, attribute-names, value, set-names.

Syntax:

ϕ: $\rightarrow$ state (empty state)

cr: entity-names $\times$ set-names $\times$ state $\rightarrow$ state

(to create a new entity)

del: entity-names $\times$ state $\rightarrow$ state

(to delete an entity)

lk: entity-names $\times$ entity-names $\times$ relation-names $\times$ state $\rightarrow$ state

(to link two entities via a given relation)

ulk: entity-names $\times$ entity-names $\times$ relation-names $\times$ state $\rightarrow$ state

(to unlink two entities in a given relation)

mod: entity-names × attribute-names × value × value × state → state

(to modify the value of an attribute of a given entity from some value to a new one)

exs: entity-names × set-names × state

(to affirm the existence of a given entity)

isr: entity-names × entity-names × relation names × state

(to affirm the relationship of two entities via a given relation)

hv: entity-names × attribute-names × value × state

(to affirm the value of a given attribute of an entity)

*: → value

(a "don't care" or unspecified value)

≈: state × state

(the equality relation on states)

The variables used in our axioms are as follows:

variables x,y,z,w: entity-names

s: state

r,r1: relation-names

u,u1: attribute-names

v,vo,v1,v2,v3: value

t,t1: set-names

We now present our axioms by first of all relating each of our predicates to the update operations and then relating each update operation to the other update operations. For example, the following axioms determine the behaviour of the predicate exs by defining the effect of each update

operation (including the constant operation ϕ).

$\neg exs(x,t,\phi)$

$\neg exs(x,t,del(x,s))$

$exs(x,t,cr(x,t,s))$

$exs(x,t,s) \Rightarrow exs(x,t,cr(y,tl,s))$

$x \neq y \wedge \neg exs(x,t,s) \Rightarrow exs(x,t,cr(y,tl,s))$

$exs(x,t,s) \Leftrightarrow exs(x,t,lk(y,w,r,s))$

$exs(x,t,s) \Leftrightarrow exs(x,t,ulk(y,w,r,s))$

$exs(x,t,s) \Leftrightarrow exs(x,t,mod(y,u,v,vl,s))$

The following are the axioms for isr:

$\neg isr(x,y,r,\phi)$

$isr(x,y,r,lk(x,y,r,s))$

$\neg isr(x,y,r,ulk(x,y,r,s))$

$isr(x,y,r,s) \Rightarrow isr(x,y,r,cr(w,t,s))$

$w \neq x \wedge w \neq y \wedge isr(x,y,r,s) \Leftrightarrow isr(x,y,r,del(w,s))$

$(x \neq z \vee y \neq w \vee r \neq rl) \wedge isr(x,y,r,s)$

$\Leftrightarrow isr(x,y,r,ulk(z,w,rl,s))$

$isr(x,y,r,s) \Rightarrow isr(x,y,r,lk(z,w,rl,s))$

$(x \neq z \vee y \neq w \vee r \neq rl) \wedge \neg isr(x,y,r,s) \Rightarrow \neg isr(x,y,r,lk(z,w,rl,s))$

$isr(x,y,r,s) \Leftrightarrow isr(x,y,r,mod(z,u,v,vl,s))$

The following are the axioms for hv:

$\neg hv(x,u,v,\phi)$

$hv(x,u,*,cr(x,t,s))$

$vl \neq v \wedge hv(x,u,v,s) \Rightarrow \neg hv(x,u,vl,s)$

$$\neg hv(x,u,v,del(x,s))$$

$$x \neq y \wedge hv(x,u,v,s) \Leftrightarrow hv(x,u,v,cr(y,t,s))$$

$$x \neq y \wedge hv(x,u,v,s) \Leftrightarrow hv(x,u,v,del(y,s))$$

$$hv(x,u,v,s) \Leftrightarrow hv(x,u,v,lk(y,z,r,s))$$

$$hv(x,u,v,s) \Leftrightarrow hv(x,u,v,ulk(y,z,r,s))$$

$$hv(x,u,v,mod(x,u,vl,v,s))$$

$$(x \neq y \vee u \neq ul) \wedge hv(x,u,v,s) \Leftrightarrow hv(x,u,v,mod(y,ul,vo,vl,s))$$

The following axioms define $\underline{\approx}$ (equivalence among states). This is done by taking each update operation in turn and relating it to each of the update operations:

$$del(x,cr(x,t,s)) \approx s$$

$$x \neq y \Rightarrow del(x,del(y,s)) \approx del(y,del(x,s))$$

$$x \neq y \wedge x \neq z \Rightarrow del(x,lk(y,z,r,s)) \approx lk(y,z,r,del(x,s))$$

$$x \neq y \wedge x \neq z \Rightarrow del(x,ulk(y,z,r,s)) \approx ulk(y,z,r,del(x,s))$$

$$x \neq y \Rightarrow del(x,mod(y,u,v,vl,s)) \approx mod(y,u,v,vl,del(x,s))$$

$$cr(x,t,cr(x,t,s)) \approx cr(x,t,s)$$

$$x \neq y \Rightarrow cr(x,t,cr(y,tl,s)) \approx cr(y,tl,cr(x,t,s))$$

$$x \neq y \Rightarrow cr(x,t,del(y,s)) \approx del(y,cr(x,t,s))$$

$$x \neq y \wedge x \neq z \Rightarrow cr(x,t,lk(y,z,r,s)) \approx lk(y,z,r,cr(x,t,s))$$

$$x \neq y \wedge x \neq z \Rightarrow cr(x,t,ulk(y,z,r,s)) \approx ulk(y,z,r,cr(x,t,s))$$

$$x \neq y \Rightarrow cr(x,t,mod(y,u,v,vl,s)) \approx mod(y,u,v,vl,cr(x,t,s))$$

$$ulk(x,y,r,lk(x,y,r,s)) \approx s$$

$$x \neq z \vee y \neq w \vee r \neq rl \Rightarrow lk(x,y,r,lk(z,w,rl,s)) \approx lk(z,w,rl,lk(x,y,r,s))$$

$$x \neq z \vee y \neq w \vee r \neq r1 \Rightarrow lk(x,y,r,ulk(z,w,r1,s)) \approx ulk(z,w,r1,lk(x,y,r,s))$$

$$lk(x,y,r,mod(z,u,v,v1,s)) \approx mod(z,u,v,v1,lk(x,y,r,s))$$

$$x \neq z \vee y \neq w \vee r \neq r1 \Rightarrow ulk(x,y,r,ulk(z,w,r1,s)) \approx ulk(z,w,r1,ulk(x,y,r,s))$$

$$ulk(x,y,r,mod(z,u,v,v1,s)) \approx mod(z,u,v,v1,ulk(x,y,r,s))$$

$$x \neq y \vee u \neq u1 \Rightarrow mod(x,u,v,v1,mod(y,u1,v2,v3,s)) \approx mod(y,u1,v2,v3,mod(x,u,v,v1,s))$$

(Note that, although we have not made the attempt here, the axioms could probably be stated as Horn clauses and an efficient language such as Prolog could be used to implement our theorem prover. See [VAN] for more details. Moreover, a unique minimal model exists for a given set of Horn clauses (again see [VAN]) and so such a set of axiom could describe a unique object (up to isomorphism) to be implemented. This is very much in the spirit of abstract data types as described in [ADJ].)

4. Updates and queries

In this section, the important notions of update and query are analysed in the context of the formalism described in the previous sections. As we shall show, in our formalism queries and updates are treated uniformly in the sense that they are expressed in the same way and that the process of answering a query is the same as that of determining a transaction which implements an update; i.e., a theorem proving process.

In the examples of updates included in this section, we will consider a slightly modified version of the axioms given in the previous sections. The reason for the modified axioms is to impose elementary integrity constraints on updates. Note, however, that queries can still be processed using the original axioms since constraints apply only to updates. This may be desirable since the original axioms are simpler (and so more efficiently processed) and it is possible to do so since we assume updates are obtained only via the stated axioms. Thus no "error" states can be generated. (At this point we might also point out that we do not need "error axioms" ([ADJ]) since we never allow the generation of error states in data bases.)

The modified axioms are:

$\neg\, exs(x,t,s) \Rightarrow \underline{exs(x,t,cr(x,t,s))}$;

$exs(x,t,s) \Rightarrow \neg\, \underline{exs(x,t,del(x,s))}$;

$exs(x,t_1,s) \wedge exs(w,t_2,s) \Rightarrow \underline{isr(x,w,r,lk(x,w,r,s))}$;

where the underlined part of the axioms are the original axioms being modified.

Now, with these new axioms we may formulate some update and query requests, starting with the empty state and incrementally updating and querying the data base. The resolution method will be used in the examples and we assume that the theorem prover incorporates some knowledge about

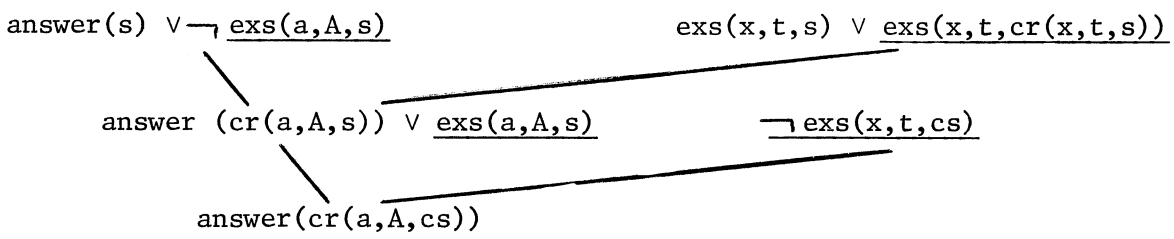
this specific problem, and that clauses instantiated to the current state will have higher priority to be resolved in each step.

In order to obtain some answer (negative or affirmative), the system may try to prove, in parallel, the theorem and its negation until one of the proofs terminates, thus giving the answer. Since we know in advance the answer to each one of our example requests, we will prove only the theorem or its negation, according to the known answer.

Example 1 (update)

"Bring into existence an entity a from entity set A"

t: $(\exists s)$ exs(a,A,s) cs = ϕ



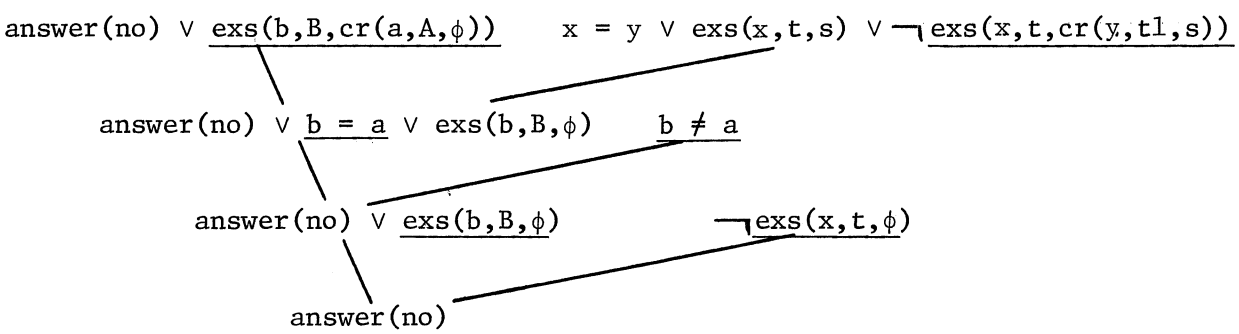
The desired state is s = cr(a,A,ϕ).

Also: exs(a,A,cr(a,A,ϕ)).

Example 2 (query)

"Does an entity b from entity set B exist?"

t: exs(b,B,cs) cs = cr(a,A,ϕ)

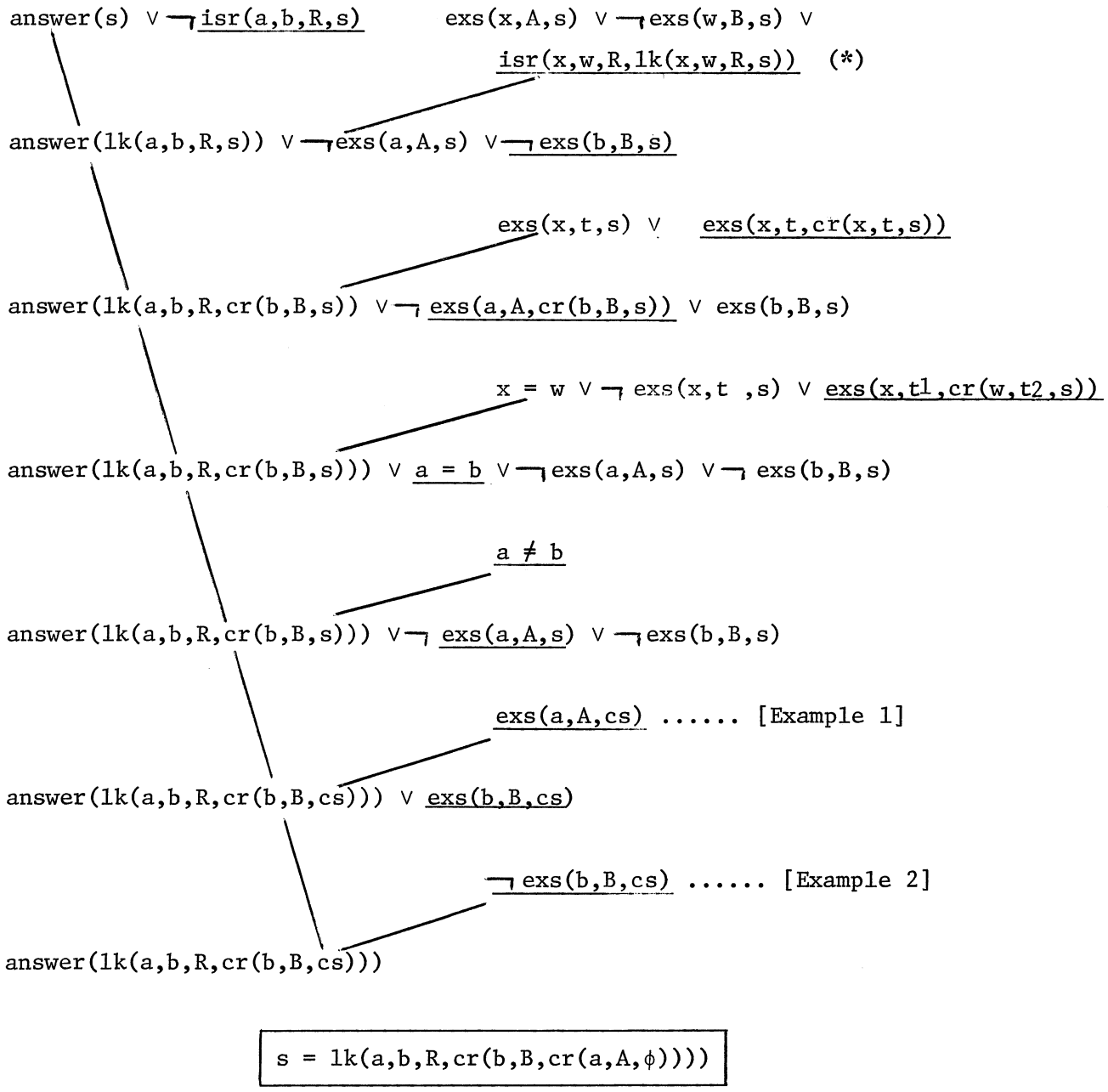


$\sim$ exs(b,B,cs)

Example 3 (update)

"Make entities a and b related via relationship R"

$(\exists s)\ isr(a,b,R,s) \qquad cs = cr(a,A,\phi)$



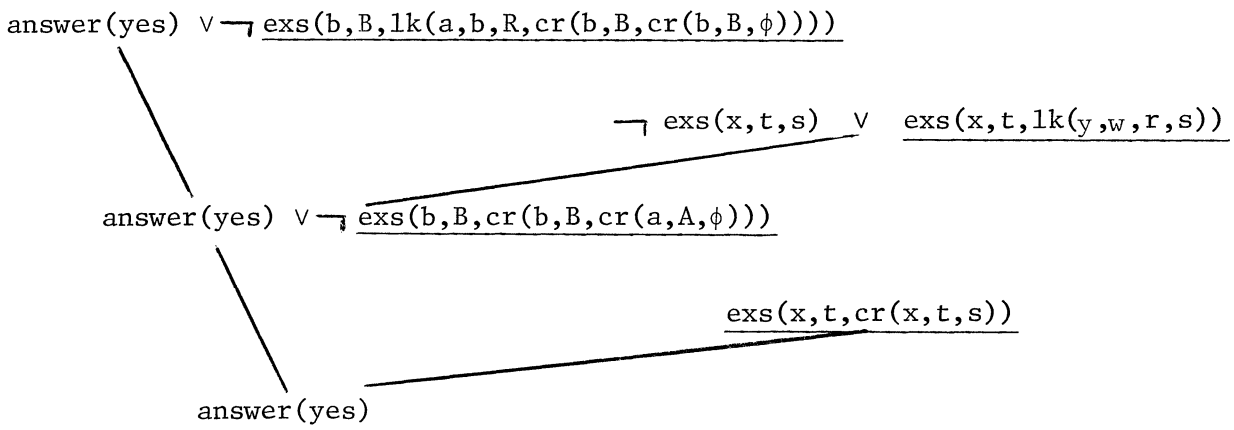
$$\boxed{s = lk(a,b,R,cr(b,B,cr(a,A,\phi))))}$$

(*) This axiom was explicitly formulated for the relationship set R, which is defined between the entity sets A and B.

Example 4 (query)

"Does an entity b from entity set B exist?"

exs(b,B,cs) cs = lk(a,b,R,cr(b,B,cr(a,A,ϕ)))

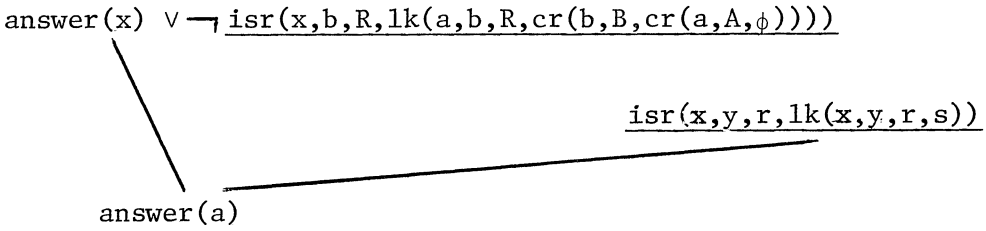


exs(b,B,cs)

Example 5 (query)

"Which is the entity (or one of the entities) related to b via R?"

(∃x) isr(x,b,R,cs) cs = lk(a,b,R,cr(b,B,cr(a,A,ϕ)))



isr(a,b,R,cs)

5. Conclusions

In the present paper, a many sorted first order predicate calculus was used as a data base specification language and it was shown how this allows a formal uniform treatment of data base queries and updates. Such a treatment can be summarized as follows. Consider initially a user's request as an open formula, where the state variable s is left free. To transform the request into a closed formula we may either:

a. substitute a state constant for s, in which case the request will be treated as a query;

or b. make s an existentially quantified variable, in which case the request will be treated as an update.

In both cases the request is handled by the system, which applies the theorem prover to the request (theorem) using the axiom set. For queries the system works like an acceptor whereas it works like a generator for updates [BOO].

At present the formalism is being used at the conceptual schema level. Future work may consider its extension for studying the mapping [ANS] between the conceptual schema and some internal schema capable of implementing the former, and between the conceptual schema and the external schemata of the various users. This would correspond to using the system like a transducer [BOO], translating objects at one level into corresponding objects at other levels.

6. References

[ADJ] J.A. Goguen, J.W. Thatcher, E.G. Wagner - An Initial Algebra Approach to the Specification, Correctness, and Implementation of Abstract Data Types, in "Current Trends in Programming Methodology, Vol. IV", ed. R.T. Yeh, Prentice-Hall, 1978.

[ANS] ANSI/X3/SPARC Study Group on Data Base Management Systems: Interim Report, FDT (Bulletin of ACM SIGMOD) 7, No. 2, 1975.

[BOO] T.L. Booth - "Sequential Machines and Automata Theory", John Wiley, 1967.

[CHE] P. Chen - The Entity-Relationship Model - Toward a Unified View of Data, ACM-TODS, Vol. 1, No. 1, 1976.

[MAI] T.S.E. Maibaum - Abstract Data Types, Database Instances and Database Modeling, submitted for publication.

[NIL] N.J. Nilsson - "Problem-solving Methods in Artificial Intelligence", McGraw-Hill, 1971.

[ROB] J.A. Robinson - A Machine Oriented Logic Based on the Resolution Principle, JACM, Vol. 12, No. 1, 1965.

[SAN] C.S. Santos, A.L. Furtado - Synthesis of Update Transactions, Technical Report TR DB107901, Depto. de Informatica, PUC/RJ, 1979.

[VAN] M.H. van Emden, R.A. Kowalski - The Semantics of Predicate Logic as a Programming Language, JACM, Vol. 23, pp. 733-742, 1976.