PUC

Series: Monografias em Ciência da Computação
Nº 14/81

CONCEPTUAL MODELLING OF DATA BASE OPERATIONS

by

C.S. Santos
T.S.E. Maibaum
A.L. Furtado

Departamento de Informática

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO
RUA MARQUÊS DE SÃO VICENTE, 225 – CEP-22453
RIO DE JANEIRO – BRASIL

PUC/RJ - Departamento de Informática

Series: Monografias em Ciência da Computação
Nº 14/81

Editor: Marco Antonio Casanova December, 1981

CONCEPTUAL MODELLING OF DATA BASE OPERATIONS*

by

C.S. Santos**
T.S.E. Maibaum***
A.L. Furtado

*Work partially sponsored by FINEP
**Universidade Federal do Rio Grande do Sul, Porto Alegre, RS-Brasil
***University of Waterloo, Waterloo, Ontario - Canada

Resumo

Um formalismo adequado para a especificação de propriedades de comportamento de bancos de dados é proposto. O formalismo é um cálculo de predicados de primeira ordem poli-sortido, incluindo uma noção formalizada de estado de banco de dados. Tanto os pedidos de atualização quanto os de consulta são modelados através de expressões pelo uso de predicados supridos na linguagem do sistema formal, e são tratados uniformemente como um processo de prova de teoremas. O processo consiste em usar os axiomas definindo o banco de dados quer para sintetizar uma sequência válida de operações de atualização (se alguma existir) quer para responder à consulta.

Palavras chaves

Bancos de dados, modelo entidades-relacionamentos, operações de atualização, operações de consulta, síntese de programas, prova de teoremas, resolução, abordagem espaço de estados, especificação axiomática de bancos de dados, tipos abstratos de dados, lógica poli-sortida.

ABSTRACT

A formalism adequate for the specification of behavioral properties of data bases is proposed. The formalism is a many-sorted first-order predicate calculus, including a formalized notion of data base state. Both update and query requests are modelled through expressions by the use of predicates supplied in the language of the formal system, and are treated uniformly as a theorem proving process. The process consists of using the axioms defining the data base for either synthesizing a valid sequence of update operations (if such exists) or for answering the query.

Keywords

Data bases, entity relationship model, update operations, query operations, program synthesis, theorem-proving, resolution, state-space approach, axiomatic specification of data bases, abstract data types, many sorted logic.

Aknowledgments

The authors are grateful to M.A. Casanova for helpful comments and suggestions. Financial support was provided by the Conselho Nacional de Desenvolvimento Científico e Tecnológico, the Canadian International Development Agency and the Financiadora de Estudos e Projetos.

1. INTRODUCTION

Current research [BR2] distinguishes two classes of properties of data bases: structural and behavioral properties. Behavioral properties are usually defined by means of procedural, executable data manipulation language statements, although there is a growing recognition [KAH] of the importance of specifying behavior at a level removed from the idiosyncrasies of particular systems and more accessible to end-users.

In this paper we propose a formalism for data base specification at the conceptual schema level [ANS], which gives an axiomatic characterization of the semantic integrity constraints governing the behavior of particular data bases. The formalism is based on the well-known state-space approach (see, for example [NIL]) and uses a many-sorted first-order predicate calculus notation.

States are formalized as compositions of update operations starting from an initial empty state. Predicates are used for asserting the existence, relatedness and attribute-values of entities [CHE,SAN] at a state. In turn, update operations are functions applied to objects and states to produce new states, and constraints are embedded in axioms expressing the applicability conditions and the intended or triggered effects of operations.

A user's request for an update is modelled by a formula (containing no direct reference to update operations) using the predicates whose truth-value the user wishes to be (or to become) true. A theorem prover would then synthesize a sequence of update operations able to move the data base from whatever is its current state to an adequate target state. (Note that some other predicates not included in the user's request may have their truth value modified so that the target state will be a valid one). 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 given set of axioms.

Conversely, the data base being in some current state, a user's request again modelled by a formula, might query the data base in the sense of:

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

b. finding for each existentially quantified variable some constant, if any, for which the formula is true.

Updates and queries are uniformly modelled 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).

With the formalism defined in this paper not only it is possible to achieve representation-free specification with high degree of portability and flexibility of implementation but also the process of verifying its correctness [BR1] (in the sense of compatibility, non-redundancy and some form of completeness) is facilitated and can be, at least partially, automated.

The formalism can also serve as a tool for simulating the use of a specified data base, for helping in the design of transactions and, in an implementation with a theorem prover, as a highly non-procedural user interface for small experimental data base systems [MI1] , where the implementation and procedural details need to be known only by the system and not by the user.

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 generalization of (one-sorted) first-order logic.

Let S be a set of sorts [MI2,REI] (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 $\langle D,B,G,R\rangle$ 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 functions so that for each $g\in G$,

$g: D_{s_1} \times \ldots \times D_{s_n} \to D_s$ 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, and 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 $(S,C,\langle F,\alpha\rangle,\langle P,\beta\rangle)$ 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 $a: F \to S^* \times S$ 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 $\beta: P \to S^*$ 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 <u>non-logical</u> 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 <u>variables of sort s</u> with

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

(ii) the equality predicate symbol $\overset{\sim}{=}_s$ of type ss.

The definitions 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. i.e., the set of variables used in formulae is also sorted.

Given L as above, a structure $D = (\Delta, \Gamma, \Phi, R)$ for L is a structure so that Δ is a family of non-empty sets sorted by S, $\Gamma = \{\Gamma_s\}_{s \in S}$ so that for each $c \in C_s$ there is $c_D \in \Gamma s$, for each $f \in F$ of type $\langle s_1 \ldots s_n, s \rangle$ there is $f_D \in \Phi$ such that $f_D: \Delta_{s_1} \times \ldots \times \Delta_{s_n} \to \Delta_s$, and for each $r \in P$ of type $s_1 \ldots s_n$ there is $r_D \in R$ such that $r_D \subseteq \Delta_{s_1} \times \ldots \times \Delta_{s_n}$.

(Note that if one wants to have more complicated sort structures, as, for example, in requiring that one sort is a subset of another, this can be acomplished by formally defining maps and axioms which define this "subsetting" or by using overloaded symbols.)

The definitions of such concepts as satisfaction, truth, etc. follow the usual one-sorted pattern. For example, given a formula ψ, we say ψ is satisfied in a structure D if for all assignements of values in Δ (the family of sets used in defining the structure) to the variables of ψ, the result is true. A structure $\mathcal{D}$ is a model of a formula ψ if $\mathcal{D}$ satisfies ψ. A similar definition holds for a set of formulae.

In general, many sorted logic can be generalized straightforwardly and without loss of any results to the many sorted case [END]. For example, we can define a deductive calculus analogous to the one-sorted logic and prove the usual completeness theorem. Other basic results such as compactness, the Lowenheim-Skolem theorem, etc., still hold for the many-sorted case.

Now consider a class of structures M over some language L, all having the same set of objects, constants, and functions. Thus they differ only in their relations. Since a relation is just a set of tuples (i.e., those satisfying the definition of the relation), we can define the structure $\cap$M which has the same objects, constants and functions as any structure in M but for each relation symbol $r \in L$, its interpretation in $\cap$ M is $\cap \{r^{\mathcal{D}} \mid \mathcal{D} \in M\}$. This clearly defines a structure over L. If each $\mathcal{D} \in M$ is a model of a set of formulae Γ, it is not always the case that $\cap$ M is a model of Γ. This does turn out to be the case if Γ is a set of Horn clauses (see below). If M is the class of Herbrand models (i.e., structures whose objects are terms over the constant and function symbols of the underlying language and whose functions are just the "expression formers") of some Γ and $\cap$M is a model of Γ, we call $\cap\Gamma$ the minimum (Herbrand) model of Γ.

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, based directly on the data base specification.

In our formalism, a data base is specified as a set $\underline{\alpha}$ 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 $\underline{t}$ (expressing a query or an update) is proved with respect to $\underline{\alpha}$.

For example, if the theorem

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

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

On the other hand, if the theorem

$$t_2: isr(a,b,R,cs),$$

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

"are entities $\underline{a}$ and $\underline{b}$ 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) predicated symbol (called a literal) with zero or more arguments, which may or may not be instantiated.

At each step of the resolution method (an iterative method), two clauses $(A \vee x)$ and $(B \vee y)$ are unified, being transformed into $(A' \vee z)$ and $(B' \vee \neg z)$, respectively, by the application of substituion rules preserving satisfiability, and then 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, a theory β expressed by the following set of axioms:

a1) Human (x) $\Rightarrow$ Male (x) $\vee$ Female (x)
a2) Male (x) $\Rightarrow$ Human (x)
a3) Female (x) $\Rightarrow$ Human (x)
a4) Male (x) $\Leftrightarrow \neg$ Female (x)
a5) Female (Mary)

and the corresponding set α of clauses:

c1) $\neg$ Human (x) $\vee$ Male (x) $\vee$ Female (x)
c2) $\neg$ Male (x) $\vee$ Human (x)
c3) $\neg$ Female (x) $\vee$ Human (x)
c4) $\neg$ Male (x) $\vee \neg$ Female (x)
c5) Male (x) $\vee$ Female (x)
c6) Female (Mary)

Now consider a candidate theorem t: Human (Mary) to be proved with respect to α. The refutation tree for this example, starting with the negation of the candidate theorem is presented in Figure 1, showing that Human (Mary) is a logical consequence of α (and also of β).

Figure 1 Here

3. DATABASE AXIOMATIZATION

We present in this section the axiomatization of a simple data base using 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 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, values, set-names.

Syntax:

$\phi: \rightarrow$ state (empty state)

cr: entity-names x set-names x state $\rightarrow$ state
(to create a new entity)

del: entity-names x state $\rightarrow$ state
(to delete an entity)

lk: entity-names x entity-names x relation-names x state $\rightarrow$ state
(to link two entities via a given relation)

ulk: entity-names x entity-names x relation-names x state $\rightarrow$ state
(to unlink two entities in a given relation)

mod: entity-names x attribute-names x value x value x state → state
(to modify the value of an attribute of a given entity from some value to a new one)

exs: entity-names x set-names x state
(to affirm the existences of a given entity)

isr: entity-names x entity-names x relation names x state
(to affirm the relationship of two entities via a given relation)

hv: entity-names x attribute-names x value x state
(to affirm the value of a given attribute of an entity)

* : → value
(a "dont't care" or unspecified value)

$\approx$: state x state
(the equality relation on states)

Thus the language $L_{ER} = (S_{ER}, C_{ER}, \langle F_{ER}, \alpha_{ER}\rangle, \langle P_{ER}, \beta_{ER}\rangle)$ of our example database is defined by:

(a) S_{ER} consists of the six sorts used above for variable declarations;

(b) $C_{ER,\ \underline{state}} = \{\phi\}$, * is in $C_{ER,\ \underline{value}}$ and, otherwise, we assume that the constants, operations and predicates for the sorts other than state are previously given. Thus the allowed entity-names, relation-names, etc., are not explicitly included in our specification of L_{ER}. If they were, the example would become unnecessarily large and its main point obscured;

(c) F_{ER} consists of cr, del, lk, ulk, mod with the types specified as above. Thus α_{ER} (cr) = <entity-names, set-names, state>, state;

(d) P_{ER} consists of exs, isr, hv, and $\approx$ with the types specified as above. Thus β ($\approx$) = <state,state>.

The variables used in our axioms are as follows:

variables x,y,z,w: entity-names
s: state
r,rl: relation-names
u,ul: attribute-names
v,v0,v1,v2,v3: value
t,tl: set-names

To have a good understanding of the data base axiomatization, the reader should be clear about the notion of data base state. A data base state may be seen as an instantaneous configuration of the data base, represented in a formal way by a composition of operations able to "construct" such a configuration from the initial (empty) state.

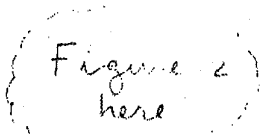
s and s' being state variables ([GRE]), we may have for example

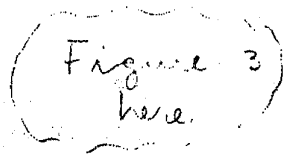
$$s = lk(b,c,l,cr(c,C,del(a,cr(b,B,cr(a,A,\phi))))),$$
$$s' = lk(b,c,l,cr(c,C,cr(b,B,\phi)))$$

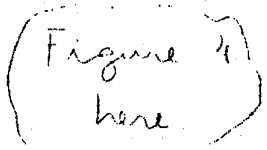
i.e. the value of s (and s') is a state in which the entities b and c exist and are related via the relationship l. Note that s and s' are equivalent states.

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 axioms in Figure 2 determine the behaviour of the predicate exs by defining the effect of each update operation (including the constant operation ϕ).

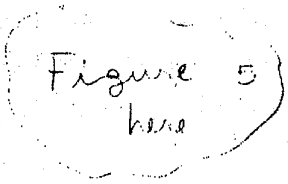
The first axiom, for example, asserts that no entity exists at the initial state and the last three assert that the existence of any entity is not affected by the operations lk, ulk, or mod. In Figures 3 and 4, the axioms for the predicates isr and hv, respectively, are presented.







The axioms presented in Figure 5 define $\approx$ (equivalence among states). This is done by taking each update operation in turn and relating it to each of the update operations. The second axiom, for example, asserts that the state resulting from the sucessive deletion of two distinct entities is independent of the order in which they are deleted.



Note that all of our axioms are or can be stated as Horn clauses, i.e. clauses containing at most one positive literal, extended with the concept of "negation as failure" [CLA], and in such case an efficient language such as PROLOG [ROU,WAR] can be used to implement the theorem prover. See [VAN] for more details. Moreover, a unique minimal model exists for a given consistent set of Horn clauses (again see ([VAN,CLA]) and so such a set of axioms could describe a unique object (up to isomorphism) to be implemented. This is very much in the spirit of abstract data types as describe in [ADJ]. Although most data bases have this property, non-Horn data bases also are reported, as in [REI].

4. AN APPLICATION

As noted in the Introduction, 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.

For updates, we will also take integrity constraints into consideration, our approach being that exactly the states and state transitions obeying the integrity constraints will be allowed.

Let us take a data base (or a view of a larger one) with two entity-sets M and O, and a binary relationship ar defined on them. It is required that an M-entity (an entity from entity-set M) can be related to at most one O-entity at each state; formally:

$$isr(x,y,ar,s) \wedge isr(x,z,ar,s) \Rightarrow y=z$$

This constraint is comparable to CODASYL's [MAN] (optional) membership, with M-entities as members and O-entities as owners.

In order to enforce a constraint such as this one we impose applicability conditions to update operations. In fact, the applicability conditions will also enforce certain obvious constraints, which we denominate natural constraints; for example, two entities can be linked only if they both exist and are not already linked via the intended relationship.

First we introduce the defined predicate isolr, which expresses the fact that a given entity is isolated with respect to a relationship, i.e. is not related to any entity via the relationship:

$$isolr(x,ar,s) \Longleftrightarrow \neg \exists\, y\ isr(x,y,ar,s)$$

The applicability conditions appear embedded in the antecedents of the axioms below:

$$exs(x,M,s) \wedge exs(y,O,s) \wedge isolr(x,ar,s) \Rightarrow isr(x,y,ar,lk(x,y,ar,s))$$

$$isr(x,y,ar,s) \Rightarrow isolr(x,ar,ulk(x,y,ar,s))$$

With the axioms previously introduced and these new axioms we may formulate update and query requests. The resolution method will be used in the examples and we assume that the theorem prover incorporates some knowledge about the specific problem, and that clauses instantiated to the current state (denoted by cs) will have higher priority to be resolved at each step.

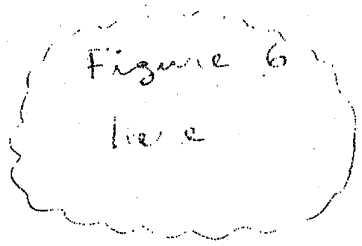
Assume that, the current state being

$$cs = cr(c,0,lk(a,b,ar,cr(a,M,cr(b,0,\phi))))$$

we wish to find which entity (if any) is related to entity a via ar. This query can be expressed as a theorem:

$$t: (\exists y)\ isr(a,y,ar,cs)$$

The refutation tree showing that b is currently related to a via ar is presented in Figure 6, where we use a special answer predicate ([GRE]) in order to get the result.



As an example of an update assume that, starting at the same current state, we wish to make a related to c via ar. The theorem is:

$$t: (\exists s)\ isr(a,c,ar,s)$$

and the refutation tree presented in Figure 7 shows that a will be related to c if we first unlink a from b and only then link a to c, thus ensuring that a will be always linked to at most one entity at a time. Notice that the three facts about the current state (grouped by braces in the refutation tree) can be effectively ascertained by the theorem-proving technique; indeed the fact

$$isr(a,b,ar,cs)$$

was obtained as the answer to our example query (as in Figure 6).

Figure 7 here

We have used axioms incorporating applicability conditions for the update requests only. For queries this is not necessary unless we wish to verify if the current state is valid. For example, if we had

$$cs = lk(a,b,cr(a,M,\phi))$$

our example query would fail if axioms with applicability conditions were used, because they require that b exists in order to be linked to a, which cannot be true if b has not been created.

However, an invalid current state can never arise as a result of updates which have been disciplined, in the ways indicated here, for effectively enforcing the declared integrity constraints.

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 the current state expression 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 can be handled by applying the theorem prover to the request (theorem) using the axioms set. For queries the system works like an acceptor whereas it works like a generator for updates [BOO]. The generation process is restricted by the specified integrity constraints.

At present the formalism is being used at the conceptual schema level. Future work may consider its extension for studying the mapping [ANS] betwenn the conceptual schema and some internal schema capable of implementing the former, and between the conceptual schema and the 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.

[BR1] M.L. Brodie, J. Schmidt - What is the Use of Abstract Data Types in Data Bases?, Proc. VLDB'4, 1978.

[BR2] M.L. Brodie - The Application of Data Types to Database Semantic Integrity, TR-833, University of Maryland, 1979.

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

[CLA] K. L. Clark - Negation as Failure, in "Logic and Data Bases", eds. H. Gallaire and J. Minker, Plenum Press, New York, 1978, pp 293-322.

[END] H.B. Enderton - "A Mathematical Introduction to Logic", Academic Press, 1972.

[GRE] C. Green - Theorem Proving by Resolution as a Basis for Question Answering Systems, in "Machine Intelligence 4", eds. B. Meltzer and D. Michie, American Elsevier Publishing Co., New York, 1969, pp 183-205.

[KAH] B. Kahn, S. Navathe, D. Smith and S. Su - Area II: Information Analysis and Definition, in "1978 New Orleans Data Base Design Workshop Report", chairman V. Lum, 1978.

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

[MAN] F. Manola - A Review of the 1978 CODASYL Database Specification, Proc. VLDB'4, 1978.

[MI1] J.Minker - Search Strategy and Selection Function for an Inferential Relational System, ACM-TODS, Vol. 3, No.1, 1978.

[MI2] J.Minker - An Experimental Relational Data Base System Based on Logic, in "Logic and Data Bases", eds. H. Gallaire and J.Minker, Plenum Press, New York, 1978, pp 107-148

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

[REI] R. Reiter - On Closed World Data Bases, in "Logic and Data Bases", eds. H. Gallaire and J. Minker, Plenum Press, New York, 1978, pp 55-76.

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

[ROU] P. Roussel - PROLOG Manuel de Reference et d'Utilization, Groupe d'Intelligence Artificielle, V.E.R., de Luminy, Université d'Aix - Marseille, Sept. 1975.

[SAN] C.S. Santos, E.J. Neuhold, A.L. Furtado - A Data Type Approach to the Entity-Relationship Model, Proc. of the Int. Conf. on the Entity - Relationship Approach to System Analysis and Design, 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.

[WAR] D. Warren - Implementing PROLOG - Compiling Predicate Logic Programs, Dept. of Artificial Intelligence, Technical Report no. 39, University of Edinburgh, 1977.

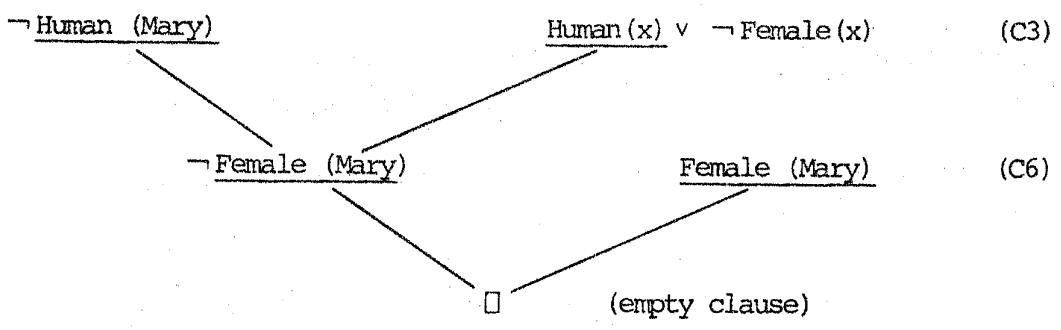


Fig. 1. Example of a refutation tree

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

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

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

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

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

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

$exs(x,t,s) \wedge x \neq y$

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

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

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

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

Fig. 2. Axioms for the exs predicate

$\neg$ isr(x,y,r,ϕ)

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

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

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

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

$\neg$ isr(x,y,r,s) $\Rightarrow$ $\neg$ 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,vl,s))

Fig. 3. Axioms for the isr predicate

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

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

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

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

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

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

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

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

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

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

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

$(x \neq y \vee u \neq ul \vee vl = v) \ \wedge \ hv(x,u,v,s) \Rightarrow hv(x,u,v,mod(y,ul,v0,vl,s))$

$(x \neq y \vee u \neq ul \vee vl \neq v) \ \wedge \ \neg hv(x,u,v,s) \Rightarrow \neg hv(x,u,v,mod(y,ul,v0,vl,s))$

Fig. 4. Axioms for the hv predicate

$$
\begin{aligned}
&del(x,cr(x,t,s)) \approx s\\
&del(x,del(y,s)) \approx del(y,del(x,s))\\
&del(x,del(x,s)) \approx del(x,s)\\
&x\neq y \wedge x\neq z \Rightarrow del(x,lk(y,z,r,s)) \approx lk(y,z,r,del(x,s))\\
&x\neq y \wedge x\neq z \Rightarrow del(x,ulk(y,z,r,s)) \approx ulk(y,z,r,del(x,s))\\
&x\neq y \Rightarrow del(x,mod(y,u,v,vl,s)) \approx mod(y,u,v,vl,del(x,s))\\
&cr(x,t,cr(x,t,cr(x,t,s))) \approx cr(x,t,s)\\
&x\neq y \Rightarrow cr(x,t,cr(y,tl,s)) \approx cr(y,tl,cr(x,t,s))\\
&x\neq y \Rightarrow cr(x,t,del(y,s)) \approx del(y,cr(x,t,s))\\
&x\neq y \wedge x\neq z \Rightarrow cr(x,t,lk(y,z,r,s)) \approx lk(y,z,r,cr(x,t,s))\\
&x\neq y \wedge x\neq z \Rightarrow cr(x,t,ulk(y,z,r,s)) \approx ulk(y,z,r,cr(x,t,s))\\
&x\neq y \Rightarrow cr(x,t,mod(y,u,v,vl,s)) \approx mod(y,u,v,vl,cr(x,t,s))\\
&lk(x,y,r,lk(x,y,r,s)) \approx lk(x,y,r,s)\\
&ulk(x,y,r,ulk(x,y,r,s)) \approx ulk(x,y,r,s)\\
&ulk(x,y,r,lk(x,y,r,s)) \approx s\\
&x\neq z \vee y\neq w \vee r\neq rl \Rightarrow lk(x,y,r,lk(z,w,rl,s))\\
&\qquad\qquad \approx lk(z,w,rl,lk(x,y,r,s))\\
&x\neq z \vee y\neq w \vee r\neq rl \Rightarrow lk(x,y,r,ulk(z,w,rl,s))\\
&\qquad\qquad \approx ulk(z,w,rl,lk(x,y,r,s))\\
&lk(x,y,r,mod(z,u,v,vl,s)) \approx\\
&\qquad\qquad mod(z,u,v,vl,lk(z,y,r,s))\\
&x\neq z \vee y\neq w \vee r\neq rl \Rightarrow ulk(x,y,r,ulk(z,w,rl,s))\\
&\qquad\qquad \approx ulk(z,w,rl,ulk(x,y,r,s))\\
&ulk(x,y,r,mod(z,u,v,vl,s)) \approx\\
&\qquad\qquad mod(z,u,v,vl,ulk(x,y,r,s))\\
&x\neq y \vee u\neq ul \Rightarrow mod(x,u,v,vl,mod(y,ul,v2,v3,s))\\
&\qquad\qquad \approx mod(y,ul,v2,v3,mod(x,u,v,vl,s))\\
&mod(x,u,vl,v2,mod(x,u,v,vl,s)) \approx mod(x,u,v,v2,s)\\
&mod(x,u,v,v,s) \approx s\\
&del(x,\phi) \approx \phi\\
&ulk(x,y,r,\phi) \approx \phi\\
&lk(x,y,r,\phi) \approx \phi
\end{aligned}
$$

Fig. 5. Axioms for the equivalence among states

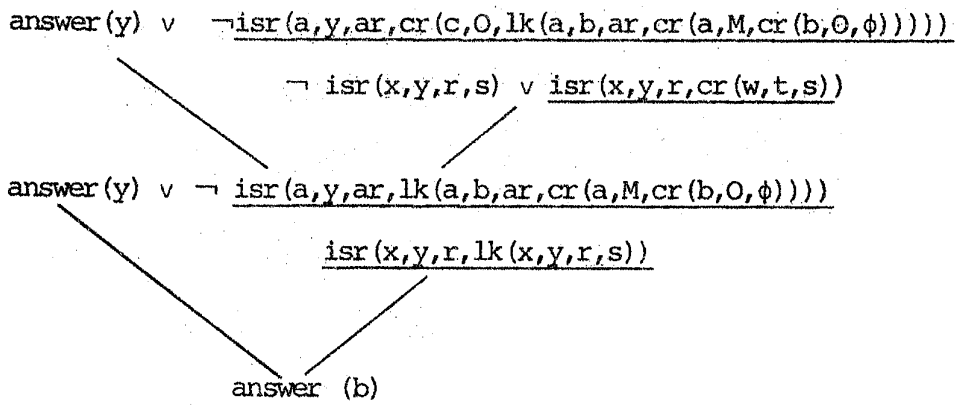


Fig. 6. Refutation tree showing that b is currently related to a via ar.

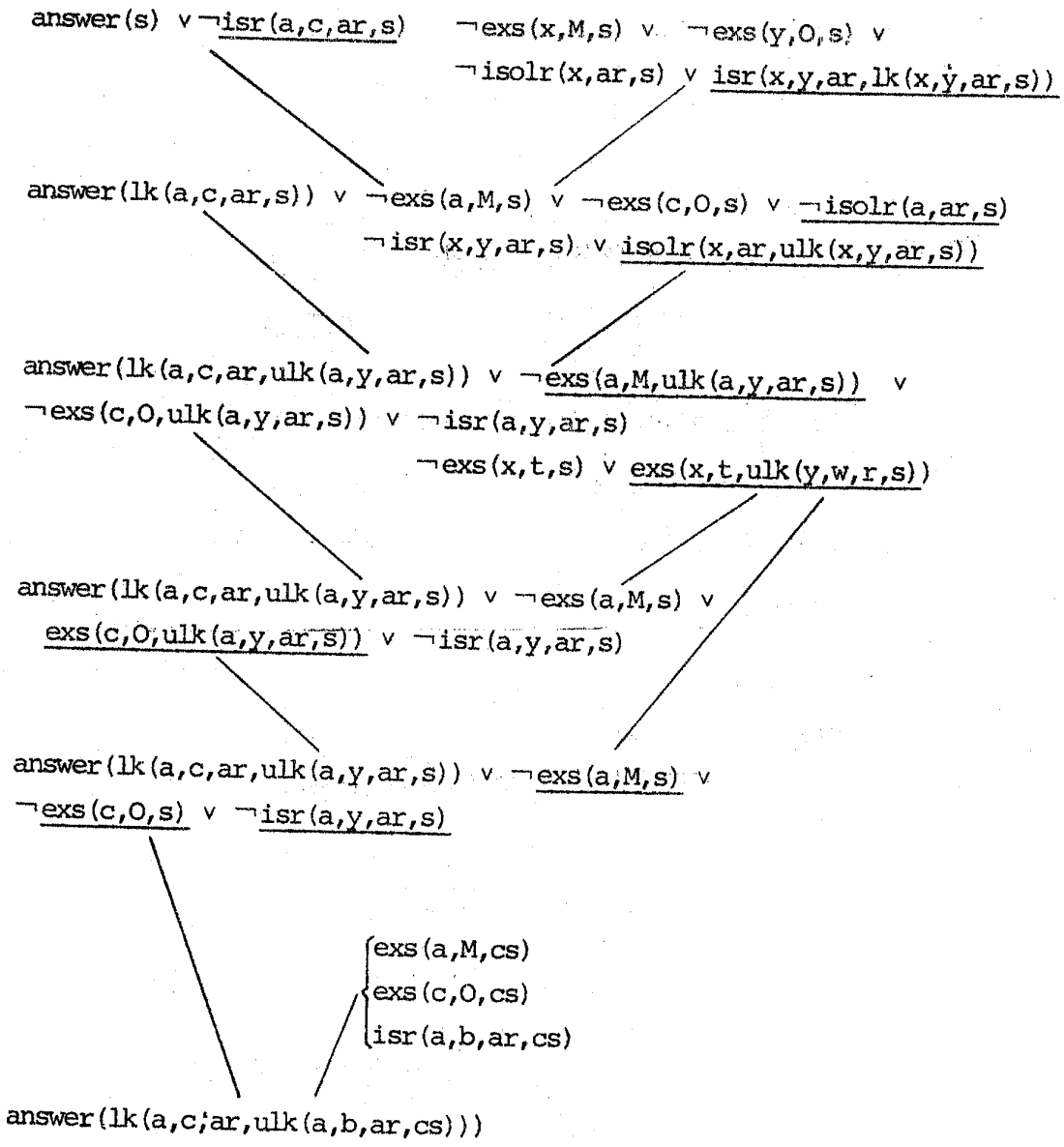


Fig. 7. Refutation tree for an update operation