On Formalizing Database Updates: Preliminary Report

Raymond Reiter

Department of Computer Science and the Canadian Institute for Advanced Research, University of Toronto, Toronto, Ont. M5S 1A4, Canada

Abstract

We address the problem of formalizing the evolution of a database under the effect of an arbitrary sequence of update transactions. We do so by appealing to a first order representation language called the situation calculus, which is a standard approach in artificial intelligence to the formalization of planning problems. We formalize database transactions in exactly the same way as actions in the artificial intelligence planning domain. This leads to a database version of the frame problem in artificial intelligence. We provide a solution to the frame problem for a special, but substantial, class of update transactions.

We next briefly describe some of the results obtained within this axiomatization. Specifically, we provide procedures for determining whether a given sequence of update transactions is legal, and for query evaluation in an updated database. These procedures have the nice property that they appeal to theorem-proving only with respect to the initial database state. We also address the problem of proving properties true in all states of the database. It turns out that mathematical induction is required for this task, and we formulate a number of suitable induction axioms. Among those properties of database states that we wish to prove are the standard database notions of static and dynamic integrity constraints. In our setting, these emerge as inductive entailments of the database.

1 Introduction

Our concern in this paper is with formalizing the evolution of a database under arbitrary sequences of update transactions. A wide variety of proposals for this exist in the literature (e.g. Grahne [4], Katsuno and Mendelzon [8], Winslett [17], Fagin, Ullman and Vardi [2], Ginsberg and Smith [3], Guessoum and Lloyd [5, 6], Manchanda and Warren [9]). In this paper, we advance a substantially different proposal.

To begin, we take seriously the fact that, during the course of its evolution, a database will pass through different states; accordingly, we endow updatable database relations with an explicit state argument which records the sequence of update transactions which the database has undergone thus far. Secondly, in our approach, the transactions themselves are first class objects, so for example, if the database admits a transaction for changing the grade g of a student st to a new grade g' for the course c, then the first order term $change(st, c, g, g')$ will be an object in the database language. These two features – an explicit state argument for updatable relations, and first order terms for transactions – are the basic ingredients of the *situation calculus*, one of the standard approaches in artificial intelligence to the formalization of planning problems. The essence of our proposal is to specify databases and their update transactions within the situation calculus.

One difficulty with this proposal, which arises immediately, is the so-called *frame problem*, well known in the planning domain. Briefly, this is the problem of how to succinctly represent the invariants of the domain, namely, those relations whose truth values are unaffected by a transaction. Thus, in the example of a grade-changing transaction, it would be necessary to state that the transaction does not affect a teacher's salary. Section 2 describes the problem in more detail, while Sections 3 and 4 describe our axiomatization of databases and transactions, and how these address the frame problem. Finally, in Section 5, we briefly describe our principal results in this approach to a theory of updates.

2 Preliminaries: The Situation Calculus and the Frame Problem

The *situation calculus* (McCarthy [10]) is a first order language designed to represent dynamically changing worlds in which all such changes are the result of named *actions*. The world is conceived as being in some state s, and this state can change only in consequence of some agent (human, robot, nature) performing an action. If α is some such action, then the successor state to s resulting from the performance of action α is denoted by $do(\alpha, s)$. In general, actions may be parameterized. For example, $put(x, y)$ might stand for the action of putting object x on object y, in which case $do(put(A, B), s)$ denotes that state resulting from placing A on B when the world is in state s. Notice that in this language, actions are denoted by function symbols. Those relations whose truth values may vary from state to state are called *fluents*, and are denoted by predicate symbols taking a state term as one of their arguments. For example, in a world in which it is possible to paint objects, we would expect a fluent $colour(x, c, s)$, meaning that the colour of object x is c when the world is in state s.

Normally, actions will have *preconditions*, namely, sufficient conditions which the current world state must satisfy before the action can be performed in this state. For example, it is possible for a robot r to pick up an object x in the world state s provided the robot is not holding any object, it is next to x, and x is not heavy:

$$[(\forall z)\neg holding(r, z, s)] \land \neg heavy(x) \land nexto(r, x, s) \supset Poss(pickup(r, x), s).^{1}$$

It is possible for a robot to repair an object provided the object is broken, and there is glue available:

$$hasglue(r, s) \land broken(x, s) \supset Poss(repair(r, x), s).$$

[1] In the sequel, lower case roman letters will denote variables. All formulas are understood to be implicitly universally quantified with respect to their free variables whenever explicit quantifiers are not indicated.

The dynamics of a world are specified by *effect axioms* which specify the effect of a given action on the truth value of a given fluent. For example, the effect of a robot dropping an object on the fluent *broken* can be specified by:

$$Poss(drop(r, x), s) \wedge fragile(x) \supset broken(x, do(drop(r, x), s)).$$

A robot repairing an object causes it not to be broken:

$$Poss(repair(r, x), s) \supset \neg broken(x, do(repair(r, x), s)).$$

As has been long recognized (McCarthy and Hayes [11]), axioms other than effect axioms are required for formalizing dynamic worlds. These are called *frame axioms*, and they specify the action *invariants* of the domain, i.e., those fluents unaffected by the performance of an action. For example, dropping things does not affect an object's colour:

$$Poss(drop(r, x), s) \wedge colour(y, c, s) \supset colour(y, c, do(drop(r, x), s)).$$

Not breaking things:

$$Poss(drop(r, x), s) \wedge \neg broken(y, s) \wedge [y \neq x \vee \neg fragile(y)] \supset \neg broken(y, do(drop(r, x), s)).$$

The problem associated with the need for frame axioms is that normally there will be a vast number of them. For example, an object's colour remains unchanged as a result of picking things up, opening a door, turning on a light, electing a new prime minister of Canada, etc. etc. Normally, only relatively few actions in any repertoir of actions about a world will affect the truth value of a given fluent; all other actions leave the fluent invariant, and will give rise to frame axioms, one for each such action. This is the *frame problem*.

In this paper, we shall propose specifying databases and update transactions within the situation calculus. Transactions will be treated exactly as actions are in dynamic worlds, i.e. they will be functions. Thus, for example, the transaction of changing a student's grade in an education database will be treated no differently than the action of dropping an object in the physical world. This means that we immediately confront the frame problem; we must find some convenient way of stating, for example, that a student's grade is unaffected by registering another student in a course, or by changing someone's address or telephone number or student number, etc. etc.

While the frame problem has been recognized in the setting of database transaction processing, notably by Borgida, Mylopoulos and Schmidt [1], it has not received any systematic treatment in the database literature. Neither is its importance widely recognized. The fact is, no adequate theory of database evolution will be possible without confronting the frame problem head-on. The next section provides an example of our approach to specifying database update transactions, and how it deals with the frame problem.

3 The Basic Approach: An Example

We consider a toy education database to illustrate our approach to specifying update transactions.

Relations The database involves the following three relations:

1. $enrolled(st, course, s)$: Student st is enrolled in course $course$ when the database is in state s.
2. $grade(st, course, grade, s)$: The grade of student st in course $course$ is $grade$ when the database is in state s.
3. $prerequ(pre, course)$: pre is a prerequisite course for course $course$. Notice that this relation is state independent, so is not expected to change during the evolution of the database.

Initial Database State We assume given some first order specification of what is true of the initial state S_0 of the database. These will be arbitrary first order sentences, the only restriction being that those predicates which mention a state, mention only the initial state S_0. Examples of information which might be true in the initial state are:

$$enrolled(Sue, C100, S_0) \lor enrolled(Sue, C200, S_0),$$
$$(\exists c) enrolled(Bill, c, S_0),$$
$$(\forall p).prerequ(p, P300) \equiv p = P100 \lor p = M100,$$
$$(\forall p) \neg prerequ(p, C100),$$
$$(\forall c).enrolled(Bill, c, S_0) \equiv c = M100 \lor c = C100 \lor c = P200,$$
$$enrolled(Mary, C100, S_0), \quad \neg enrolled(John, M200, S_0), \ldots$$
$$grade(Sue, P300, 75, S_0), \quad grade(Bill, M200, 70, S_0), \ldots$$
$$prerequ(M200, M100), \quad \neg prerequ(M100, C100), \ldots$$

Database Transactions Update transactions will be denoted by function symbols, and will be treated in exactly the same way as actions are in the situation calculus. For our example, there will be three transactions:

1. $register(st, course)$: Register student st in course $course$.
2. $change(st, course, grade)$: Change the current grade of student st in course $course$ to $grade$.
3. $drop(st, course)$: Student st drops course $course$.

Transaction Preconditions Normally, transactions have preconditions which must be satisfied by the current database state before the transaction can be "executed". In our example, we shall require that a student can register in a course iff she has obtained a grade of at least 50 in all prerequisites for the course:

$$Poss(register(st, c), s) \equiv \{(\forall p).prerequ(p, c) \supset (\exists g).grade(st, p, g, s) \land g \geq 50\}.$$

It is possible to change a student's grade iff he has a grade which is different than the new grade:

$$Poss(change(st, c, g), s) \equiv (\exists g').grade(st, c, g', s) \land g' \neq g.$$

A student may drop a course iff the student is currently enrolled in that course:

$$Poss(drop(st, c), s) \equiv enrolled(st, c, s).$$

Update Specifications These are the central axioms in our formalization of update transactions. They specify the effects of all transactions on all updatable database relations. As usual, all lower case roman letters are variables which are implicitly universally quantified. In particular, notice that these axioms quantify over transactions.

$$\begin{aligned} &Poss(a, s) \supset \\ &\quad [enrolled(st, c, do(a, s)) \equiv \\ &\qquad a = register(st, c) \vee enrolled(st, c, s) \wedge a \neq drop(st, c)], \end{aligned} \tag{1}$$

$$\begin{aligned} &Poss(a, s) \supset \\ &\quad [grade(st, c, g, do(a, s)) \equiv \\ &\qquad a = change(st, c, g) \vee grade(st, c, g, s) \wedge (\forall g')a \neq change(st, c, g')]. \end{aligned}$$

It is the update specification axioms which "solve" the frame problem. To see why, let α be *any* transaction distinct from $register(st, c)$ and $drop(st, c)$. We obtain the following instance of the axiom (1):

$$Poss(\alpha, s) \supset \{enrolled(st, c, do(\alpha, s)) \equiv enrolled(st, c, s)\},$$ [2]

i.e., $register(st, c)$ and $drop(st, c)$ are the only transactions which can possibly affect the truth value of $enrolled$; *all other transactions leave its truth value unchanged* (provided $Poss(\alpha, s)$ is true, of course).[3] But this ability to succinctly represent all of the transactions which leave a given fluent invariant is precisely the kind of solution to the frame problem that we seek. A little reflection reveals those properties of the axiom (1) which solve the problem for us:

1. Quantification over transactions, and
2. The assumption that relatively few transactions (in this case $register(st, c)$ and $drop(st, c)$) affect the truth value of the fluent, so that the sentence (1) is reasonably short. In other words, most transactions leave a fluent's truth value unchanged, which of course is what originally lead to too many frame axioms.

4 An Axiomatization of Updates

The example education domain illustrates the general principles behind our approach to the specification of database update transactions. In this section we precisely characterize a class of databases and updates of which the above example will be an instance.

[2] Notice that to obtain this instance we require unique names axioms for transactions, i.e.,

$$change(st, c, g) \neq drop(st, c),$$
$$drop(st, c) \neq register(st, c),$$
$$etc.$$

[3] Since for our example there are just three transactions, this might not seem much of an achievement. To see that it is, simply imagine augmenting the set of transactions with arbitrarily many new transactions, each of which is irrelevant to the truth of *enrolled*, say transactions for changing students' registration numbers, addresses, telephone numbers, fees, etc. etc.

Unique Names Axioms for Transactions For distinct transaction names T and T',

$$T(\mathbf{x}) \neq T'(\mathbf{y})$$

Identical transactions have identical arguments:

$$T(x_1, ..., x_n) = T(y_1, ..., y_n) \supset x_1 = y_1 \wedge ... \wedge x_n = y_n$$

for each function symbol T of $\mathcal{L}$ of sort *transaction*.

Unique Names Axioms for States

$$(\forall a, s) S_0 \neq do(a, s),$$

$$(\forall a, s, a', s').do(a, s) = do(a', s') \supset a = a' \wedge s = s'.$$

Definition: The Simple Formulas The *simple* formulas are defined to be the smallest set such that:

1. $F(\mathbf{t}, s)$ and $F(\mathbf{t}, S_0)$ are simple whenever F is a fluent, the $\mathbf{t}$ are terms, and s is a variable of sort *state* .[4]
2. Any equality atom is simple. Notice that equality atoms, unlike fluents, are permitted to mention the function symbol *do*.
3. Any other atom with predicate symbol other than *Poss* is simple.
4. If S_1 and S_2 are simple, so are $\neg S_1$, $S_1 \wedge S_2$, $S_1 \vee S_2$, $S_1 \supset S_2$, $S_1 \equiv S_2$.
5. If S is simple, so are $(\exists x)S$ and $(\forall x)S$ whenever x is an individual variable not of sort *state*.

In short, the simple formulas are those first order formulas which mention only domain predicate symbols, whose fluents do not mention the function symbol *do*, and which do not quantify over variables of sort *state*.

Definition: Transaction Precondition Axiom A transaction precondition axiom is a formula of the form

$$(\forall \mathbf{x}, s).Poss(T(x_1, \cdots, x_n), s) \equiv \Pi_T,$$

where T is an n-ary transaction function, and Π_T is a simple formula whose free variables are among $x_1, \cdots, x_n, s$.

Definition: Successor State Axiom A successor state axiom for an $(n+1)$-ary fluent F is a sentence of the form

$$(\forall a, s).Poss(a, s) \supset (\forall x_1, \ldots, x_n).F(x_1, \ldots, x_n, do(a, s)) \equiv \Phi_F$$

where, for notational convenience, we assume that F's last argument is of sort *state*, and where Φ_F is a simple formula, all of whose free variables are among $a, s, x_1, \ldots, x_n$.

[4] For notational convenience, we assume that the last argument of a fluent is always the (only) argument of sort *state*.

5 Results

We here briefly indicate some of the results we have obtained for first order databases axiomatized as in the previous section. The full details may be found in (Reiter [15]).

5.1 Legal Transaction Sequences

Not all transaction sequences need be legal. For example, the sequence

$$drop(Sue, C100), change(Bill, C100, 60)$$

would be illegal if the *drop* transaction was impossible in the initial database state, i.e. if $Poss(drop(Sue, C100), S_0)$ was false. Even if the *drop* transaction were possible, the sequence would be illegal if the *change* transaction was impossible in that state resulting from doing the *drop* transaction, i.e. if

$$Poss(change(Bill, C100, 60), do(drop(Sue, C100), S_0))$$

was false.

Intuitively, a transaction sequence is legal iff, beginning in state S_0, each transaction in the sequence is possible in that state resulting from performing all the transactions preceeding it in the sequence. Our concern is to characterize the legal transaction sequences. To that end, we require the following:

Definition: A Regression Operator $\mathcal{R}$ Let W be first order formula. Then $\mathcal{R}[W]$ is that formula obtained from W by replacing each fluent atom $F(\mathbf{t}, do(\alpha, \sigma))$ mentioned by W by $\Phi_F(\mathbf{t}, \alpha, \sigma)$ where F's successor state axiom is

$$(\forall a, s).Poss(a, s) \supset (\forall \mathbf{x}).F(\mathbf{x}, do(a, s)) \equiv \Phi_F(\mathbf{x}, a, s).$$

All other atoms of W not of this form remain the same.

Regression corresponds to the operation of *unfolding* in logic programming.

In what follows, $\mathcal{D}_{uns}$ and $\mathcal{D}_{unt}$ denote the unique names axioms for states and unique names axioms for transactions, respectively (Section 4). $\mathcal{D}_{S_0}$ denotes the initial database, i.e. any set of sentences which mention only the initial state term S_0. (See Section 3 for an example $\mathcal{D}_{S_0}$.)

Theorem 1. *Let $\tau_1, \ldots, \tau_n$ be a sequence of update transactions. There is a formula $R_{\tau_1,\ldots,\tau_n}$ (which is easily obtained using the regression operator and the transaction precondition axioms of Section 3) such that $\tau_1, \ldots, \tau_n$ is legal wrt $\mathcal{D}$ iff*

$$\mathcal{D}_{uns} \cup \mathcal{D}_{unt} \cup \mathcal{D}_{S_0} \models R_{\tau_1,\ldots,\tau_n}.[5]$$

Theorem 1 informs us that *legality testing reduces to first order theorem proving in the initial database $\mathcal{D}_{S_0}$ together with unique names axioms.*

[5] See (Reiter [15]) for a description of how to compute $R_{\tau_1,\ldots,\tau_n}$.

5.2 Query Evaluation

Notice that in our formalization of update transactions in the situation calculus, all updates are *virtual*; the database is never physically changed. In order to query a database resulting from a sequence of update transactions we must explicitly refer to this sequence in the query. For example, to ask whether John is enrolled in any courses after the transaction sequence $drop(John, C100)$, $register(Mary, C100)$ has been 'executed', we must determine whether

$$Database \models (\exists c).enrolled(John, c, do(register(Mary, C100), do(drop(John, C100), S_0))).$$

In general, the specific problem we address is this: Given a sequence $\tau_1, \ldots, \tau_n$ of transactions, and a query $Q(s)$ whose only free variable is the state variable s, what is the answer to Q in that state resulting from performing this transaction sequence, beginning with the initial database state S_0? This can be formally defined as the problem of determining whether

$$\mathcal{D} \models Q(do([\tau_1, \ldots, \tau_n], S_0)),$$

where $\mathcal{D}$ denotes some database theory. Here, $do([\tau_1, \ldots, \tau_n], S_0)$ abbreviates the state term

$$do(\tau_n, do(\tau_{n-1}, \cdots, do(\tau_1, S_0) \cdots)),$$

and denotes that state resulting from performing the transaction τ_1, followed by τ_2, ..., followed by τ_n, beginning with the initial database state S_0.

Our principal result is the following:

Theorem 2. (Soundness and Completeness of Query Evaluation) *Suppose that $\tau_1, \ldots, \tau_n$ is a legal transaction sequence. Then whenever $\mathcal{D}$ is a database whose axioms are those of Section 4,*

$$\mathcal{D} \models Q(do([\tau_1, \ldots, \tau_n], S_0))$$

iff

$$\mathcal{D}_{uns} \cup \mathcal{D}_{unt} \cup \mathcal{D}_{S_0} \models \mathcal{R}^n[Q(do([\tau_1, \ldots, \tau_n], S_0))].$$ [6]

As was the case for legality testing, *query evaluation reduces to first order theorem proving in the initial database $\mathcal{D}_{S_0}$ together with unique names axioms.*

5.3 Integrity Constraints

In database theory, an *integrity constraint* specifies what counts as a legal database state; it is a property that every database state must satisfy. In our setting, it is natural to represent these as first order sentences, universally quantified over states. In the following examples, $\leq$ is a binary relation between states; $s \leq s'$ means that s' is a possible future of s, i.e. there is some sequence of zero or more transactions which can lead a database from state s to state s'. As it happens, defining $\leq$ requires a second order axiom, the details of which we omit here. A definition is given in (Reiter [15]).

[6] Recall that $\mathcal{D}_{uns}$ and $\mathcal{D}_{unt}$ denote the unique names axioms for states and unique names axioms for transactionss, respectively (Section 4), and $\mathcal{D}_{S_0}$ denotes the axioms true of the initial database state.

Examples of integrity constraints

1. No one may have two different grades for the same course in any database state. This is a standard functional dependency.

$$(\forall s)(\forall st, c, g, g').S_0 \leq s \wedge grade(st, c, g, s) \wedge grade(st, c, g', s) \supset g = g'.$$

2. Salaries must never decrease. This is the classic example of a *dynamic* integrity constraint.

$$(\forall s, s')(\forall p, \$, \$').S_0 \leq s \wedge s \leq s' \wedge sal(p, \$, s) \wedge sal(p, \$', s') \supset \$ \leq \$'.$$

Constraint satisfaction defined: A database *satisfies* an integrity constraint IC iff

$$Database \models IC.$$[7]

Not surprisingly, some form of mathematical induction is necessary to establish that a database satisfies an integrity constraint. In [15], we justify the following second order axiom of induction suitable for proving properties of states s in the situation calculus, whenever $S_0 \leq s$.

$$\begin{aligned}(\forall W).W(S_0) \wedge [(\forall a, s).Poss(a, s) \wedge S_0 \leq s \wedge W(s) \supset W(do(a, s))] \\ \supset (\forall s).S_0 \leq s \supset W(s).\end{aligned}$$

This is our analogue of the standard second order induction axiom for Peano arithmetic.

Frequently, we want to prove sentences of the form

$$(\forall s, s').S_0 \leq s \wedge s \leq s' \supset R(s, s').$$

For example, the following classic dynamic integrity constraint has this form:

$$(\forall s, s', p, \$, \$').S_0 \leq s \wedge s \leq s' \supset sal(p, \$, s) \wedge sal(p, \$', s') \supset \$ \leq \$'.$$

Using the above simple induction axiom we can derive an induction axiom suitable for proving properties of pairs of states s and s' when $S_0 \leq s \wedge s \leq s'$:

$$\begin{aligned}&(\forall R).R(S_0, S_0) \wedge \\ &[(\forall a, s).Poss(a, s) \wedge S_0 \leq s \supset R(s, do(a, s))] \wedge \\ &[(\forall a, s, s').Poss(a, s) \wedge S_0 \leq s \wedge R(s, s) \supset R(do(a, s), do(a, s))] \wedge \\ &[(\forall a, s, s').Poss(a, s') \wedge S_0 \leq s \wedge s \leq s' \wedge R(s, s') \supset R(s, do(a, s'))] \\ &\qquad \supset (\forall s, s').S_0 \leq s \wedge s \leq s' \supset R(s, s').\end{aligned}$$

(Reiter [15]) provides a number of examples of integrity constraints and their verification using these induction axioms.

[7] This definition should be contrasted with those in Reiter [14, 13]. It seems that there is not a unitary concept of integrity constraint in database theory, and that there are many subtleties involved.

6 Conclusions and Extensions

Database transactions have long been treated procedurally. We have outlined a declarative treatment of this notion by appealing to suitable axiomatizations in the situation calculus. A number of issues have not been addressed in this paper; these include the following:[8]

- **Historical Queries:** On our account of database evolution, databases are never physically modified and therefore never forget. It is therefore possible to pose and answer *historical queries*, for example "Did Mary ever get a raise?"
- **Logic Programming Implementation:** It seems that our approach to database updates can be implemented in a fairly straightforward way as a logic program, thereby directly complementing the logic programming perspective on databases (Minker [12]).
- **Indeterminate Transactions:** A limitation of our formalism is that it requires all transactions to be *determinate*, by which we mean that in the presence of complete information about the initial database state a transaction completely determines the resulting state. An example of an indeterminate transaction is registering a student in a multi-section course, without specifying in the parameters of the transaction, in which of the possible sections the student is registered. It is possible to extend the theory of this paper to include indeterminate transactions by appealing to a simple idea for dealing with the frame problem due to Haas [7], as elaborated by Schubert [16].

Acknowledgments Many of my colleagues provided important conceptual and technical advice. My thanks to Leo Bertossi, Alex Borgida, Craig Boutilier, Michael Gelfond, Gösta Grahne, Russ Greiner, Joe Halpern, Hector Levesque, Vladimir Lifschitz, Fangzhen Lin, Wiktor Marek, John McCarthy, Alberto Mendelzon, John Mylopoulos, Javier Pinto, Len Schubert, Yoav Shoham and Marianne Winslett. Funding for this work was provided by the National Science and Engineering Research Council of Canada, and by the Institute for Robotics and Intelligent Systems.

References

1. A. Borgida, J. Mylopoulos, and J. Schmidt. The TaxisDL software description language. Technical report, Department of Computer Science, University of Toronto, 1991.
2. R. Fagin, J.D. Ullman, and M.Y. Vardi. Updating logical databases. In *Proceedings of the ACM Symposium on Principles of Database Systems*, April 1983.
3. M.L. Ginsberg and D.E. Smith. Reasoning about actions I: A possible worlds approach. *Artificial Intelligence*, 35:165–195, 1988.
4. G. Grahne. Updates and counterfactuals. In J. Allen, R. Fikes, and E. Sandewall, editors, *Proceedings of the Second International Conference on Principles of Knowledge Representation and Reasoning (KR'91)*, pages 269–276, Los Altos, CA, 1991. Morgan Kaufmann Publishers, Inc.
5. A. Guessoum and J.W. Lloyd. Updating knowledge bases. *New Generation Computing*, 8(1):71–89, 1990.

[8] See (Reiter [15]) for more details.

6. A. Guessoum and J.W. Lloyd. Updating knowledge bases II. Technical report, University of Bristol, 1991. To appear.
7. A. R. Haas. The case for domain-specific frame axioms. In F. M. Brown, editor, *The frame problem in artificial intelligence. Proceedings of the 1987 workshop*, pages 343–348, Los Altos, California, 1987. Morgan Kaufmann Publishers, Inc.
8. H. Katsuno and A.O. Mendelzon. On the difference between updating a knowledge base and revising it. In J. Allen, R. Fikes, and E. Sandewall, editors, *Proceedings of the Second International Conference on Principles of Knowledge Representation and Reasoning (KR'91)*, pages 387–394, Los Altos, CA, 1991. Morgan Kaufmann Publishers, Inc.
9. S. Manchanda and D.S. Warren. A logic-based language for database updates. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 363–394. Morgan Kaufmann Publishers, Inc., Los Altos, CA, 1988.
10. J. McCarthy. Programs with common sense. In M. Minsky, editor, *Semantic Information Processing*, pages 403–418. The MIT Press, Cambridge, MA, 1968.
11. J. McCarthy and P. Hayes. Some philosophical problems from the standpoint of artificial intelligence. In B. Meltzer and D. Michie, editors, *Machine Intelligence 4*, pages 463–502. Edinburgh University Press, Edinburgh, Scotland, 1969.
12. J. Minker, editor. *Foundations of Deductive Databases and Logic Programming.* Morgan Kaufmann Publishers, Inc., Los Altos, CA, 1988.
13. R. Reiter. What should a database know? *Journal of Logic Programming.* to appear.
14. R. Reiter. Towards a logical reconstruction of relational database theory. In M.L. Brodie, J. Mylopoulos, and J.W. Schmidt, editors, *On Conceptual Modelling: Perspectives from Artificial Intelligence, Databases and Programming Languages*, pages 191–233. Springer, New York, 1984.
15. R. Reiter. On specifying database updates. Technical report, Department of Computer Science, University of Toronto, in preparation.
16. L.K. Schubert. Monotonic solution of the frame problem in the situation calculus: an efficient method for worlds with fully specified actions. In H.E. Kyberg, R.P. Loui, and G.N. Carlson, editors, *Knowledge Representation and Defeasible Reasoning*, pages 23–67. Kluwer Academic Press, 1990.
17. M. Winslett. Reasoning about action using a possible models approach. In *Proceedings of the National Conference on Artificial Intelligence*, pages 89–93. American Association for Artificial Intelligence, 1988.

This article was processed using the LaTeX macro package with LMAMULT style