The Projection Problem in the Situation Calculus: A Soundness and Completeness Result, with an Application to Database Updates

Raymond Reiter
Department of Computer Science
University of Toronto
Toronto, Canada M5S 1A4
and
The Canadian Institute for Advanced Research
email: reiter@ai.toronto.edu

Abstract

We describe a novel application of planning in the situation calculus to formalize the evolution of a database under update transactions. In the resulting theory, query evaluation becomes identical to the temporal projection problem. We next define a class of axioms for which the classical AI planning technique of goal regression provides a sound and complete method for solving the projection problem, hence for querying evolving databases. Finally, we briefly discuss several issues which naturally arise in the settings of databases and planning, namely, proofs by mathematical induction of properties of world states, logic programming implementations of the projection problem, and historical queries.

1 Introduction

The situation calculus (McCarthy [7]) is enjoying new popularity these days. One reason is that its expressiveness is considerably richer than has been commonly believed (Gelfond, Lifschitz, Rabinov [2]). Another is the possibility of precisely characterizing the strengths and limitations of various general theories of actions expressed within its formalism (see [6, 5, 12] for examples). In this paper we propose yet another reason for not giving up too hastily on the situation calculus. Specifically, we propose a novel application of it to the problem of specifying the evolution of a database under update transactions. Our proposal is to represent a database in the situation calculus. Updatable database relations will be fluents; transactions will be functions, and will be treated exactly as are actions in the usual situation calculus formalizations of dynamic worlds. As we shall see, querying an evolving database is precisely the *temporal projection problem* in AI planning. This motivates the theoretical focus of this paper, which presents a sound and complete regression-style procedure for solving the projection problem in the case of a limited, but sizable class of background axiomatizations.

The theoretical results of this paper complement those of (Reiter [10]), where a sound and complete plan synthesis procedure is described, based on goal regression. Proofs of our results, and further discussion, may be found in (Reiter [12]).

2 Formalizing Database Evolution in the Situation Calculus: An Example

In this section we describe a novel application of the situation calculus and its relationship to the temporal projection problem in planning. In the theory of databases, the evolution of a database is determined by *transactions*, whose purpose is to update the database with new information. For example, in an educational database, there might be a transaction specifically designed to change a student's grade. This would normally be a procedure which, when invoked on a specific student and grade, first checks that the database satisfies the transaction's preconditions (e.g., that there is a record for the student, and that the new grade differs from the old), and if so, records the new grade. In current database practice, transactions are procedures which physically modify data structures representing the current database state, much like STRIPS operators. Our objective in this section is to provide a *specification* of the effects of transactions on database states. Our proposal is to represent a database in the situation calculus. Updatable database relations will be fluents; transactions will be functions, and will be treated exactly as are actions in the usual situation calculus formalizations of dynamic worlds.

To illustrate our approach to specifying database transactions, we consider the following toy education database.

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:

$$enrolled(Sue, C100, S_0) \vee enrolled(Sue, C200, S_0),$$

$$(\exists c) enrolled(Bill, c, S_0),$$

$$(\forall p).prerequ(p, P300) \equiv p = P100 \vee p = M100,$$

$$(\forall p) \neg prerequ(p, C100),$$

$$(\forall c).enrolled(Bill, c, S_0) \equiv \\ c = M100 \vee c = C100 \vee c = P200,$$

$$enrolled(Mary, C100, S_0),$$

$$\neg enrolled(John, M200, S_0), \ldots$$

$$grade(Sue, P300, 75, S_0),$$

$$grade(Bill, M200, 70, S_0), \ldots$$

$$prerequ(M200, M100), \quad \neg prerequ(M100, C100), \ldots$$

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:[1]

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

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

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

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

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

Transaction 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.

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

$$Poss(a, s) \supset [grade(st, c, g, do(a, s)) \equiv \\ a = change(st, c, g) \vee \\ grade(st, c, g, s) \wedge (\forall g') a \neq change(st, c, g')].$$

Although incidental to the main thrust of this paper, notice that it is these transaction specification axioms which "solve" the frame problem in the database setting. The equivalences in these axioms were motivated by Pednault's approach to the frame problem [9], and the appeal to quantification over transactions stems from the *explanation closure axioms* of Haas [3] and Schubert [14]. See (Reiter [10, 12]) for further discussion.

Querying a Database Notice that in the above account of database evolution, all updates are *virtual*; the database is never physically changed. To query the database resulting from some sequence of transactions, it is necessary to refer to this sequence in the query. For example, to determine if John is enrolled in any courses after the transaction sequence

$$drop(John, C100), register(Mary, C100)$$

has been 'executed', we must determine whether

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

Thus, querying an evolving database is precisely the *temporal projection problem* in AI planning [4]; this motivates the theoretical focus of this paper.

[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.

3 An Axiomatization

In this section we precisely characterize the class of axioms which the theory of this paper addresses. The axioms of Section 2 all fit these patterns. Throughout, we assume a sorted second order language $\mathcal{L}$, with sorts for states and for actions.[2] We omit the details of this language, which are described in (Reiter [12]).

Unique Names Axioms for Actions For distinct action names T and T',

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

Identical actions 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 *action*.

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 of $\mathcal{L}$ are those first order formulas which do not mention the predicate symbols *Poss*, $<$ or $\leq$, whose fluents do not mention the function symbol *do*, and which do not quantify over variables of sort *state*.

Definition: Action Precondition Axiom An action precondition axiom is a sentence of the form

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

where T is an n-ary function of sort *action* of $\mathcal{L}$, and Π_T is a simple formula of $\mathcal{L}$ 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 of $\mathcal{L}$ is a sentence of $\mathcal{L}$ of the form

$$\begin{aligned}&(\forall a, s).Poss(a, s) \supset \\ &\qquad (\forall x_1, \ldots, x_n).F(x_1, \ldots, x_n, do(a, s)) \equiv \Phi_F\end{aligned}$$

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 $a, s, x_1, \ldots, x_n$.

Notice that these successor state axioms are suitable only for formalizing *determinate* actions, i.e. actions whose effects on all fluents are completely known. We could not appeal to axioms of this form for characterizing the action of putting down an object, when the resulting precise location of the object is unknown.[3] While this is a severe restriction in practice, database applications normally appeal to determinate transactions, in which case these successor state axioms are precisely what we want.

Notice also, as remarked in Section 2, these axioms deal with the frame problem. They do not, alas, address the *ramification problem* (Finger [1]), which is to say that the ramifications of any action must be explicitly represented in the successor state axioms. Recent results by Lin and Shoham [6] appear to address this problem in our setting.

4 Regression and the Projection Problem

> "In solving a problem of this sort, the grand thing is to be able to reason backward."
>
> Sherlock Holmes, *A Study in Scarlet*

This section describes our formal results for the temporal projection problem in the situation calculus. It defines a regression operator, and uses it to characterize legal plans, as well as a systematic procedure for solving the projection problem.

4.1 Legal Plans

Not all plans need be legal. Intuitively, an action sequence is legal iff, beginning in state S_0, the preconditions of each action in the sequence are true in that state resulting from performing all the actions preceeding it in the sequence. To formalize this notion of a legal plan, we first define an ordering relation $<$ on states . The intended interpretation of $s < s'$ is that state s' is reachable from state s by some sequence of actions, each action of which is possible in that state resulting from executing the actions preceeding it in the sequence. Hence, we want $<$ to be the smallest binary relation on states such that:

1. $\sigma < do(a, \sigma)$ whenever action a is possible in state σ, and
2. $\sigma < do(a, \sigma')$ whenever action a is possible in state σ' and $\sigma < \sigma'$.

This we can achieve with a second order sentence, as follows:[4]

[2] We require a second order language in order to define the concept of a legal action sequence. See Section 4.1 below.

[3] At least, we could not do so whenever a fluent $location(x, l, s)$ is part of the theory.

[4] This cannot be done with a first order sentence since it requires transitive closure, which is well known not to be first order definable.

Definitions: $s < s'$, $s \leq s'$

$$(\forall s, s').s < s' \equiv \\ (\forall P).\{[(\forall a, s_1).Poss(a, s_1) \supset P(s_1, do(a, s_1))] \wedge \\ [(\forall a, s_1, s_2).Poss(a, s_2) \wedge P(s_1, s_2) \supset \\ P(s_1, do(a, s_2))]\} \\ \supset P(s, s'). \quad (2)$$

$$(\forall s, s').s \leq s' \equiv s < s' \vee s = s'. \quad (3)$$

Intuitively, $S_0 \leq s$ states that s is an executable plan, i.e., it is a sequence of plan steps, each of whose action preconditions is true in the previous state.

Notation: Let $a_1, \ldots, a_n$ be terms of sort *action*. Define

$$do([\,], s) = s,$$

and for $n = 1, 2, \ldots$

$$do([a_1, \ldots, a_n], s) = do(a_n, do([a_1, \ldots, a_{n-1}, s)).$$

$do([a_1, \ldots, a_n], s)$ is a compact notation for the state term

$$do(a_n, do(a_{n-1}, \ldots do(a_1, s) \ldots))$$

which denotes that state resulting from performing the action a_1, followed by $a_2, \ldots$, followed by a_n, beginning in state s.

Definition: The Legal Action Sequences Suppose $\tau_1, \ldots, \tau_n$ is a sequence of ground terms (i.e. terms not mentioning any variables) of $\mathcal{L}$, where each τ_i is of sort *action*. Then this sequence is *legal (with respect to some background axiomatization $\mathcal{D}$)* iff

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

Definition: In the sequel, we shall only consider background axiomatizations $\mathcal{D}$ of the form:

$$\mathcal{D} = \textit{less-axioms} \cup \mathcal{D}_{ss} \cup \mathcal{D}_{ap} \cup \mathcal{D}_{uns} \cup \mathcal{D}_{una} \cup \mathcal{D}_{S_0} \quad (4)$$

where

- *less-axioms* are the axioms (2), (3) for $<$ and $\leq$.
- $\mathcal{D}_{ss}$ is a set of successor state axioms.
- $\mathcal{D}_{ap}$ is a set of action precondition axioms.
- $\mathcal{D}_{uns}$ is the set of unique names axioms for states.
- $\mathcal{D}_{una}$ is the set of unique names axioms for actions.
- $\mathcal{D}_{S_0}$ is a set of first order sentences with the property that S_0 is the only term of sort *state* mentioned by the fluents of a sentence of $\mathcal{D}_{S_0}$. See Section 2 for an example $\mathcal{D}_{S_0}$. Thus, no fluent of a formula of $\mathcal{D}_{S_0}$ mentions a variable of sort *state* or the function symbol *do*. $\mathcal{D}_{S_0}$ will play the role of the initial database (i.e. the one we start off with, before any actions have been "executed").

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

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

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

The use of the regression operator $\mathcal{R}$ is a classical plan synthesis technique (Waldinger [15]). Regression also corresponds to the operation of *unfolding* in logic programming.

Notation: When G is a formula of $\mathcal{L}$,

$$\mathcal{R}^0[G] = G,$$

$$\mathcal{R}^n[G] = \mathcal{R}[\mathcal{R}^{n-1}[G]] \quad n = 1, 2, \ldots$$

Recall that for each function T of sort *action*, $\mathcal{D}_{ap}$ will contain an axiom of the form

$$(\forall \vec{x}, s).Poss(T(\vec{x}), s) \equiv \Pi_T(\vec{x}, s),$$

Theorem 1 (Reiter [12]) *Let $T_1, \ldots, T_m$ be function symbols of sort action. Then the sequence $T_1(\vec{g}_1), \ldots, T_m(\vec{g}_m)$ of ground terms is legal wrt $\mathcal{D}$ iff*

$$\mathcal{D}_{uns} \cup \mathcal{D}_{una} \cup \mathcal{D}_{S_0} \models \\ \textstyle\bigwedge_{i=1}^{m} \mathcal{R}^{i-1}[\Pi_{T_i}(\vec{g}_i, do([T_1(\vec{g}_1), \ldots, T_{i-1}(\vec{g}_{i-1})], S_0))].$$

Theorem 1 reduces the test for the legality of an action sequence to a first order theorem proving task *in the initial database $\mathcal{D}_{S_0}$*, together with suitable unique names axioms.

Example: Legality Testing
In connection with the example database of Section 2, we determine the legality of the transaction sequence

$$change(Bill, C100, 60), register(Sue, C200), \\ drop(Bill, C100).$$

We first compute

$$\mathcal{R}^0[(\exists g')grade(Bill, C100, g', S_0) \wedge g' \neq 60] \wedge \\ \mathcal{R}[(\forall p)prerequ(p, C200) \supset \\ (\exists g).grade(Sue, p, g, do(change(Bill, C100, 60), S_0)) \\ \wedge g \geq 50] \wedge \\ \mathcal{R}^2[enrolled(Bill, C100, do(register(Sue, C200), \\ do(change(Bill, C100, 60), S_0)))].$$

This simplifies to

$$\{(\exists g').grade(Bill, C100, g', S_0) \wedge g' \neq 60\} \wedge \\ \{(\forall p).prerequ(p, C200) \supset Bill = Sue \wedge p = C100 \vee \\ (\exists g).grade(Sue, p, g, S_0) \wedge g \geq 50\} \wedge \\ \{Sue = Bill \wedge C200 = C100 \vee \\ enrolled(Bill, C100, S_0)\}.$$

So the transaction sequence is legal iff this formula is entailed by the initial database, together with unique names axioms for states and actions.

4.2 A Solution to the Projection Problem

The following provides a systematic regression-based procedure for solving the projection problem for the class of axioms (4) defined above.

Theorem 2 (Soundness and Completeness (Reiter [12])) *Suppose $Q(s) \in \mathcal{L}$ is simple, and that the state variable s is the only free variable of $Q(s)$. Suppose $\tau_1, \ldots, \tau_n$ is a sequence of ground terms of $\mathcal{L}$ of sort action. Then if $\tau_1, \ldots, \tau_n$ is a legal action sequence,*

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

iff

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

As in the case of verifying legality, the projection problem reduces to first order theorem proving *in the initial database* $\mathcal{D}_{S_0}$, together with unique names axioms.

Corollary 1 (Relative Consistency) *$\mathcal{D}$ is satisfiable iff $\mathcal{D}_{uns} \cup \mathcal{D}_{una} \cup \mathcal{D}_{S_0}$ is.*

Corollary 1 provides an important relative consistency result. It guarantees that we cannot introduce an inconsistency to a "base" theory $\mathcal{D}_{uns} \cup \mathcal{D}_{una} \cup \mathcal{D}_{S_0}$ by augmenting it with the axioms for $<$ and $\leq$, together with successor state and action precondition axioms.

Example: Query Evaluation
Continuing with the example education database, consider again the transaction sequence

$$\begin{aligned}\mathbf{T} = {} & change(Bill, C100, 60), register(Sue, C200), \\ & drop(Bill, C100).\end{aligned}$$

Suppose the query is

$$\begin{aligned}& (\exists st).enrolled(st, C200, do(\mathbf{T}, S_0)) \wedge \\ & \neg enrolled(st, C100, do(\mathbf{T}, S_0)) \wedge \\ & (\exists g).grade(st, C200, g, do(\mathbf{T}, S_0)) \wedge g \geq 50.\end{aligned}$$

We must compute $\mathcal{R}^3$ of this query. After some simplification, assuming that $\mathcal{D}_{S_0} \models C100 \neq C200$, we obtain

$$\begin{aligned}& (\exists st).[st = Sue \vee enrolled(st, C200, S_0)] \wedge \\ & [st = Bill \vee \neg enrolled(st, C100, S_0)] \wedge \\ & [(\exists g).grade(st, C200, g, S_0) \wedge g \geq 50].\end{aligned}$$

Therefore, assuming that the transaction sequence $\mathbf{T}$ is legal, the answer to the query is obtained by evaluating this last formula in $\mathcal{D}_{S_0}$.

5 Discussion

The striking similarities between databases evolving under update transactions and dynamically changing worlds in the situation calculus raises a number of additional issues of common interest to the database and planning communities:

- **Proving properties of world states:** In commonsense reasoning about the physical world, we often want to establish properties which will be true no matter what the state of the world is, for example, that if an object is broken and it never gets repaired, then it will always be broken:

$$\begin{aligned}& (\forall x).broken(x, S_0) \wedge \\ & \quad [(\forall s).S_0 \leq s \supset \neg occurs(repair(x), s)] \supset \\ & \quad\quad (\forall s').S_0 \leq s' \supset broken(x, s').\end{aligned}$$

  Here, $occurs(a, s)$ means that action a occurs in the sequence of actions leading from S_0 to s. An analogous problem arises in connection with *integrity constraints* in database theory. Informally, an integrity constraint specifies what counts as a legal database state; it is a property that every database state must satisfy. One of the classic examples of such a constraint is that no one's salary may decrease during the evolution of the database:

$$\begin{aligned}& (\forall s, s')(\forall p, \$, \$').S_0 \leq s \wedge s \leq s' \supset \\ & \quad sal(p, \$, s) \wedge sal(p, \$', s') \supset \$ \leq \$'.\end{aligned}$$

  As might be expected, proving such properties of states in the situation calculus requires mathematical induction. Using the axioms (2) and (3) for $<$ and $\leq$, Reiter [?, 11] derives suitable induction axioms for this task, and gives various examples of integrity constraints and their proofs.

- **Logic programming implementation:** As it happens, there is a natural translation of the axioms of Section 3 into Prolog clauses, thereby directly complementing the logic programming perspective on databases (Minker [8]). For example, the successor state axiom (1) of Section 2 is represented by two Prolog clauses:

$$\begin{aligned}& enrolled(st, c, do(a, s)) \leftarrow \\ & \quad a = register(st, c), Poss(a, s).\end{aligned}$$

$$\begin{aligned}& enrolled(st, c, do(a, s)) \leftarrow \\ & \quad a \neq drop(st, c), enrolled(st, c, s), Poss(a, s).\end{aligned}$$

  Similar clauses may be proposed for the other axioms of Section 2 (see Reiter [?, 13]). With a suitable clausal form for $\mathcal{D}_{S_0}$, it would then be possible to evaluate queries against updated databases, for example

$$\begin{aligned}\leftarrow enrolled(& John, C200, \\ & do(register(Mary, C100), \\ & \quad do(drop(John, C100), S_0))).\end{aligned}$$

  In other words, there appears to be a natural logic programming implementation of the temporal projection problem. Presumably, all of this can be made to work under suitable conditions. The remaining problem is to characterize what these conditions are, and to prove correctness of such an implementation with respect to the logical specification of this paper.

- **Historical queries:** Using the relations $<$ and $\leq$ on states, it is possible to pose *historical* queries to a database. For example, if $\mathbf{T}$ is the transaction sequence leading to the current database state (i.e., the current database state is $do(\mathbf{T}, S_0)$), the following asks whether Mary's salary was ever less than it is now:

  $$(\exists s, \$, \$').S_0 \leq s \wedge s < do(\mathbf{T}, S_0) \wedge sal(Mary, \$, s) \wedge sal(Mary, \$', do(\mathbf{T}, S_0)) \wedge \$ < \$'.$$

  Such queries are of some interest for databases. The analogous questions for planning would be something like: If this plan is executed, would such and such ever be true during the plan execution? (Will block A ever be on block B during the execution of this plan?) Within the framework of this paper, it is possible to develop a theory of such queries (Reiter [13]).

Acknowledgements

Many of my colleagues provided important conceptual and technical advice. My thanks to Leo Bertossi, Alex Borgida, Craig Boutilier, Charles Elkan, 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] J. Finger. *Exploiting Constraints in Design Synthesis.* PhD thesis, Stanford University, Stanford, CA, 1986.

[2] M. Gelfond, V. Lifschitz, and A. Rabinov. What are the limitations of the situation calculus? In *Working Notes, AAAI Spring Symposium Series on the Logical Formalization of Commonsense Reasoning*, pages 59–69, 1991.

[3] 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.

[4] S. Hanks and D. McDermott. Default reasoning, nonmonotonic logics, and the frame problem. In *Proceedings of the National Conference on Artificial Intelligence*, pages 328–333, 1986.

[5] V. Lifschitz. Toward a metatheory of action. 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 376–386, Los Altos, CA, 1991. Morgan Kaufmann Publishers, Inc.

[6] F. Lin and Y. Shoham. Provably correct theories of action. In *Proceedings of the National Conference on Artificial Intelligence*, 1991.

[7] J. McCarthy. Programs with common sense. In M. Minsky, editor, *Semantic Information Processing*, pages 403–418. The MIT Press, Cambridge, MA, 1968.

[8] J. Minker, editor. *Foundations of Deductive Databases and Logic Programming.* Morgan Kaufmann Publishers, Inc., Los Altos, CA, 1988.

[9] E.P.D. Pednault. ADL: Exploring the middle ground between STRIPS and the situation calculus. In R.J. Brachman, H. Levesque, and R. Reiter, editors, *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning (KR'89)*, pages 324–332. Morgan Kaufmann Publishers, Inc., 1989.

[10] R. Reiter. The frame problem in the situation calculus: a simple solution (sometimes) and a completeness result for goal regression. In Vladimir Lifschitz, editor, *Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy*, pages 359–380. Academic Press, San Diego, CA, 1991.

[11] R. Reiter. Proving properties of states in the situation calculus. *Artificial Intelligence*, 64:337–351, 1993.

[12] R. Reiter. A simple solution to the frame problem (sometimes). Technical report, Department of Computer Science, University of Toronto, in preparation.

[13] R. Reiter. Formalizing database evolution in the situation calculus. In *Proc. Fifth Generation Computer Systems*, pages 600–609, Tokyo, June 1 - 5, 1992.

[14] 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.

[15] R. Waldinger. Achieving several goals simultaneously. In E. Elcock and D. Michie, editors, *Machine Intelligence 8*, pages 94–136. Ellis Horwood, Edinburgh, Scotland, 1977.