ON SPECIFYING DATABASE UPDATES

RAYMOND REITER

▷ 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. Using the axioms corresponding to this solution, 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 next 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 principles. 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.

Finally, we discuss various possible extensions of the approach of this paper, including transaction logs and historical queries, the complexity of query evaluation, actualized transactions, logic programming approaches to updates, database views and state constraints. ◁

This paper consolidates and expands on a variety of results, some of which have been described elsewhere (Reiter [46, 45, 44]).

Address correspondence to Department of Computer Science
University of Toronto
Toronto, Canada M5S 1A4
email: reiter@ai.toronto.edu

THE JOURNAL OF LOGIC PROGRAMMING

655 Avenue of the Americas, New York, NY 10010 0743-1066/94/$7.00

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. Abiteboul [1], Grahne [13], Katsuno and Mendelzon [20], Winslett [48], Fagin, Ullman and Vardi [10], Ginsberg and Smith [12], Guessoum and Lloyd [16, 17], Manchanda and Warren [32], Kowalski [22], Bonner and Kifer [6].). In this paper, we advance a substantially different approach.

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 citizens, 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 individual 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 which arises immediately is the so-called *frame problem*, well known in the artificial intelligence planning literature. 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. 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.

With this axiomatization in hand, we are in a position to address query evaluation for updated databases. This we do in Section 5, where we provide procedures for determining whether a given sequence of update transactions is legal, and for querying an updated database. These procedures have the nice property that they appeal to theorem-proving only with respect to the initial database state.

In Section 6, we 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 principles. 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.

Subsequently, in Section 7, we discuss various possible extensions of the approach of this paper, including transaction logs and historical queries, the complexity of query evaluation, actualized transactions, logic programming approaches to updates, database views and state constraints.

We close with Section 8, which provides a comparative discussion of various approaches to a theory of database updates.

A pleasant consequence of our appeal to the situation calculus as a database representation language is that, in almost all respects, the resulting theory of database updates is isomorphic to the theory of planning in dynamic worlds as studied in artificial intelligence. This formal identity provides a potentially fruitful synthesis of problems and solutions from both disciplines.

2. PRELIMINARIES: THE SITUATION CALCULUS AND THE FRAME PROBLEM

The *situation calculus* (McCarthy [33]) 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:

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

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

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

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, on the fluent *broken*, of a robot dropping an object 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 [35]), 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:

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

[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. We also assume that $\wedge$ takes precedence over $\vee$, so that $a \wedge b \vee c \wedge d$ means $(a \wedge b) \vee (c \wedge d)$.

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.

The frame problem has been recognized in the setting of database transaction processing, notably by Kowalski [22] and Borgida, Mylopoulos and Schmidt [7]. It is also implicit in various semantic approaches to database updates (but without appealing explicitly to transactions) such as the work of Grahne [13], Katsuno and Mendelzon [20], Grahne, Mendelzon and Revesz ([14]) and Winslett [48]. Our approach differs from these semantic accounts in two ways: it explicitly provides for transactions, and it relies on an axiomatic treatment of the frame problem. The next section provides an example of our axiomatic approach to specifying database update transactions, and how it addresses 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:

$$(\forall x).enrolled(x, C100, S_0) \supset enrolled(x, C200, S_0),$$

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

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

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

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

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

$$enrolled(Mary, C100, S_0), \quad \neg enrolled(John, M200, S_0), \ldots$$

$$grade(Sue, P300, 75, S_0), \quad grade(Bill, M200, 70, S_0), \ldots$$

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

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:

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

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:

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

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.

$$\begin{aligned} &Poss(a, s) \supset [enrolled(st, c, do(a, s)) \equiv \\ &\qquad a = register(st, c) \vee enrolled(st, c, s) \wedge a \neq drop(st, c)], \end{aligned} \tag{3.1}$$

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

This last sentence is logically equivalent to the simpler:

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

It is the update specification axioms which "solve" the frame problem. To see why, notice that (3.1) entails:

$$Poss(a,s) \wedge a \neq register(st,c) \wedge a \neq drop(st,c) \supset \\ \{enrolled(st,c,do(a,s)) \equiv enrolled(st,c,s)\},$$

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).[2][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 (3.1) 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 (3.1) 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.

For a more detailed description of this approach to the frame problem, and a procedure for automatically obtaining this solution from the effect axioms alone, see (Reiter [40]). For an independently motivated circumscriptive justification of this solution to the frame problem, see (Lin and Reiter [29]).

3.1. 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

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

[2] Notice that to draw this conclusion 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.

has been 'executed', we must determine whether

$$\begin{array}{l} Database \models \\ \quad (\exists c).enrolled(John, c, do(register(Mary, C100), do(drop(John, C100), S_0))). \end{array}$$

Querying an evolving database is precisely what is called the *temporal projection problem* in AI planning [18].

4. AN AXIOMATIZATION OF UPDATE TRANSACTIONS

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. To begin, we must specify a second order language on which to base the axiomatization.[4]Let $\mathcal{L}$ be a sorted second order language with equality, with two disjoint sorts for transactions and states, and suppose these sorts are disjoint from any other sorts of the language. Assume $\mathcal{L}$ has the following vocabulary:

- Individual variables: Infinitely many of each sort.
- Predicate variables: infinitely many of each arity, each of which takes arguments all of which are of sort *state.*
- Function symbols of sort *state*: There are just two of these - the constant S_0, and the binary function symbol *do* which takes arguments of sort *transaction* and *state* respectively.
- Function symbols of sort *transaction*: Finitely many.
- Other function symbols: Infinitely many of sort other than *transaction* and *state* for each arity, none of which take an argument of sort *state.*
- Predicate symbols:
  1. A distinguished binary predicate symbol *Poss* taking arguments of sort *transaction* and *state* respectively.
  2. A distinguished binary predicate symbol $<$ taking arguments of sort *state.*
  3. Finitely many predicate symbols, distinct from the predicate symbols *Poss*, $<$ and $\leq$, each of which takes, among its arguments, exactly one of sort *state*; these are called *fluents.* Notice that the predicate symbols *Poss*, $<$ and $\leq$, which do take arguments of sort *state*, are not fluents.
  4. Infinitely many predicate symbols of each arity, none of which take arguments of sort *state.*
- Logical constants and punctuation: As usual, including equality.

Notice that $\mathcal{L}$ does not allow state dependent functions like *employer-of(x,s)*, or *Canadian-prime-minister(s).*

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

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

[4] The language must be second order because we shall require a transitive ordering relation, $<$, on states, and this is not first order definable (Section 5.1).

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

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

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

Notice that the unique names axioms for states imply that two states are the same iff they result from the same sequence of transactions applied to the initial state. Two states S_1 and S_2 may be different, yet assign the same truth value to all fluents. So a state in the situation calculus must not be identified with the set of fluents which hold in that state. A better way to understand a state is as a *history* of transactions; two states are equal iff they have identical histories.

Definition: The Simple Formulas The *simple* formulas of $\mathcal{L}$ are defined to be the smallest set such that:

1. $F(\vec{t}, s)$ and $F(\vec{t}, S_0)$ are simple whenever F is a fluent, the $\vec{t}$ are terms, and s is a variable of sort *state*.[5]
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* or $<$ 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 do not mention the predicate symbols *Poss* or $<$, 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 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 *transaction* 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

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

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

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

5. TRANSACTION LOGS AND QUERY EVALUATION

In many database applications, a *log* is maintained of the sequence of (virtual) update transactions that has occurred against the database, and queries are processed with respect to this log and the initial (static) database. We emphasize that these transactions are virtual; they are not *actualized* on the given initial database. Our objective in this section is to present a sound and complete query evaluator for this case. The general problem is this: Given a query Q, and a sequence $\tau_1, \ldots, \tau_n$ of update transactions, is this sequence legal, and if so, what is the answer to Q in that state of the database that would result from performing these transactions in the indicated sequence, beginning with the initial state S_0 of the database? This is exactly what is called the *temporal projection problem* in the AI planning literature ([18]). For the class of databases of this paper, Reiter [43] has provided a closed form solution to this problem, which we now describe.

5.1. Legal Transaction Sequences

In this section we provide necessary and sufficient conditions that a sequence $\tau_1, \ldots, \tau_n$ of update transactions be legal. Notice that 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. To formalize this notion, we 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 transactions, each transaction of which is possible in that state resulting from executing the transactions preceeding it in the sequence. As in (Reiter [42]), we begin by postulating the following axioms:

$$(\forall s)\neg s < S_0. \tag{5.1}$$

$$(\forall a, s, s').s < do(a, s') \equiv Poss(a, s') \wedge s \leq s'. \tag{5.2}$$

Here, $s \leq s'$ is an abbreviation for $s < s' \vee s = s'$.

In addition, we shall later need a (second order) induction axiom over states, so we include that here for future reference:

$$(\forall P).P(S_0) \wedge (\forall a, s)[P(s) \supset P(do(a, s))] \supset (\forall s)P(s). \tag{5.3}$$

Compare this with the induction axiom for the natural numbers:

$$(\forall P).P(0) \wedge (\forall x)[P(x) \supset P(succ(x))] \supset (\forall x)P(x).$$

Just as the induction axiom for the natural numbers restricts the domain of numbers to 0 and its successors, the effect of the induction axiom (5.3) is to restrict the state domain of any of its models to be isomorphic to the smallest set $\mathcal{S}$ satisfying:

1. $S_0 \in \mathcal{S}$.

2. If $S \in \mathcal{S}$, and $A \in \mathcal{A}$, then $do(A, S) \in \mathcal{S}$, where $\mathcal{A}$ is the domain of actions in the model.

Notation: $do([a_1, \ldots, a_n], s)$ Let $a_1, \ldots, a_n$ be terms of sort *transaction.* Define

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

$$do([a_1, \ldots, a_n], s) = do(a_n, do([a_1, \ldots, a_{n-1}, s)) \quad n = 1, 2, \ldots$$

$do([a_1, \ldots, a_n], s)$ 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 transaction a_1, followed by $a_2, \ldots$, followed by a_n, beginning in state s.

Definition: The Legal Transaction 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 *transaction.* Then this sequence is *legal (with respect to some background database axiomatization $\mathcal{D}$)* iff

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

Definition: Databases In the sequel, a *database* $\mathcal{D}$ will always be a set of sentences of $\mathcal{L}$ of the following form:

$$\mathcal{D} = \Sigma \cup \mathcal{D}_{ss} \cup \mathcal{D}_{tp} \cup \mathcal{D}_{uns} \cup \mathcal{D}_{unt} \cup \mathcal{D}_{S_0}$$

where

- Σ is the set consisting of the above three axioms (5.1), (5.2) and (5.3).
- $\mathcal{D}_{ss}$ is a set of successor state axioms, one for each fluent of $\mathcal{L}$.
- $\mathcal{D}_{tp}$ is a set of transaction precondition axioms, one for each transaction function of $\mathcal{L}$.
- $\mathcal{D}_{uns}$ is the set of unique names axioms for states.
- $\mathcal{D}_{unt}$ is the set of unique names axioms for transactions.
- $\mathcal{D}_{S_0}$ 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 $\mathcal{D}_{S_0}$. 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 transactions have been "executed").

Notice that the induction axiom (5.3) is the only second order sentence of $\mathcal{D}$; all other sentences of $\mathcal{D}$ are first order.

Definition: A Regression Operator We now introduce an operator corresponding to the notion of *goal regression* as it arises in artificial intelligence planning problems (Waldinger [47]). It is also a parallel version of the operation of *unfolding* in logic programming. The purpose of the regression operator is to systematically reduce the complexity of ground state terms occurring in situation calculus formulas; by repeatedly applying this operator, we eventually obtain a formula whose only state term is S_0. As the following theorems show, this reduces theorem proving for formulas with arbitrary ground state terms to theorem proving for formulas whose only state term is S_0.

Assume given a database $\mathcal{D}$, as defined above. The *regression operator* $\mathcal{R}$ when applied to a formula of $\mathcal{L}$ is determined relative to the database $\mathcal{D}$ and is defined recursively as follows:

1. When A is a non-fluent atom, including equality atoms, and atoms with predicate symbol $Poss$ or $<$,
   $\mathcal{R}[A] = A$.
2. When ϕ is a fluent atom whose state argument is a variable,
   $\mathcal{R}[\phi] = \phi$.
3. When F is a fluent whose successor state axiom in $\mathcal{D}_{ss}$ is
   $$(\forall a, s).Poss(a, s) \supset (\forall x_1, \ldots, x_n).F(x_1, \ldots, x_n, do(a, s)) \equiv \Phi_F \qquad (5.4)$$
   then
   $\mathcal{R}[F(t_1, \ldots, t_n, do(\alpha, \sigma))] = \Phi_F|^{x_1,\ldots,x_n,a,s}_{t_1,\ldots,t_n,\alpha,\sigma}$.
4. Whenever W is a formula,
   $\mathcal{R}[\neg W] = \neg\mathcal{R}[W]$,
   $\mathcal{R}[(\forall v)W] = (\forall v)\mathcal{R}[W]$,
   $\mathcal{R}[(\exists v)W] = (\exists v)\mathcal{R}[W]$.
5. Whenever W_1 and W_2 are formulas,
   $\mathcal{R}[W_1 \wedge W_2] = \mathcal{R}[W_1] \wedge \mathcal{R}[W_2]$,
   $\mathcal{R}[W_1 \vee W_2] = \mathcal{R}[W_1] \vee \mathcal{R}[W_2]$,
   $\mathcal{R}[W_1 \supset W_2] = \mathcal{R}[W_1] \supset \mathcal{R}[W_2]$,
   $\mathcal{R}[W_1 \equiv W_2] = \mathcal{R}[W_1] \equiv \mathcal{R}[W_2]$.

$\mathcal{R}[G]$ is simply that formula obtained from G by substituting suitable instances of Φ_F in F's successor state axiom for each occurrence in G of a fluent atom of the form $F(t_1, \ldots, t_n, do(\alpha, \sigma))$.

Example

$$\begin{aligned} G = (\forall a, s).P(A, do(a, do(a', s))) \wedge s = do(B, S_0) \supset \\ (\exists x).P(x, s) \wedge Poss(B, do(a, s)) \wedge R(x) \wedge Q(do(B, s)). \end{aligned}$$

Here P and Q are fluents; R is not a fluent. Suppose the successor state axioms for P and Q are

$$Poss(a, s) \supset [P(x, do(a, s)) \equiv \Phi_P(x, a, s)],$$

$$Poss(a, s) \supset [Q(do(a, s)) \equiv \Phi_Q(a, s)].$$

Then

$$\begin{aligned} \mathcal{R}[G] = (\forall a, s).\Phi_P(A, a, do(a', s)) \wedge s = do(B, S_0) \supset \\ (\exists x).P(x, s) \wedge Poss(B, do(a, s)) \wedge R(x) \wedge \Phi_Q(B, s). \end{aligned}$$

The idea behind the regression operator $\mathcal{R}$ is to reduce the depth of nesting of the function symbol do in the fluents of G by substituting suitable instances of Φ_F from (5.4) for each occurrence of a fluent atom of G of the form $F(t_1, \ldots, t_n, do(\alpha, \sigma))$. Since no fluent atom of Φ_F mentions the function symbol do, the effect of this substitution is to replace each such F by a formula whose fluents mention only the state term σ, and this reduces the depth of nesting by one.

Definition: $\mathcal{R}^n$ When G is a formula of $\mathcal{L}$,

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

For $n = 1, 2, \ldots$

$$\mathcal{R}^n[G] = \mathcal{R}[\mathcal{R}^{n-1}[G]].$$

Suppose τ is a ground transaction term, say $T(g_1, \ldots, g_k)$, and suppose T's transaction precondition axiom is:

$$(\forall x_1, \ldots, x_k, s).Poss(T(x_1, \ldots, x_k), s) \equiv \Pi_T(x_1, \ldots, x_k, s).$$

Define $precond(\tau, s)$ to be the formula $\Pi_T(g_1, \ldots, g_k, s)$. The formula $precond(\tau, s)$ specifies the conditions under which the ground transaction τ is possible in state s.

The following is proved in (Reiter [43]):

Theorem 5.1. The sequence $\tau_1, \ldots, \tau_n$ of ground terms of $\mathcal{L}$ of sort transaction is legal wrt $\mathcal{D}$ iff

$$\mathcal{D}_{unt} \cup \mathcal{D}_{S_0} \models \bigwedge_{i=1}^{n} \mathcal{R}^{i-1}[precond(\tau_i, do([\tau_1, \ldots, \tau_{i-1}], S_0))].$$

Notice that Theorem 5.1 reduces the test for the legality of a transaction sequence to a first order theorem proving task *in the initial database* $\mathcal{D}_{S_0}$, together with unique names axioms for transactions. In particular, the second order induction axiom is not required for the purpose of testing legality.

Example: Legality Testing We compute the legality test for the transaction sequence

$$register(Bill, C100), drop(Bill, C100), drop(Bill, C100)$$

which intuitively should fail because the first *drop* leaves *Bill* unenrolled in $C100$, so that the precondition for the second *drop* will be false. We must first compute

$\mathcal{R}^0[precond(register(Bill, C100), S_0)] \wedge$
$\mathcal{R}[precond(drop(Bill, C100), do(register(Bill, C100), S_0))] \wedge$
$\mathcal{R}^2[precond(drop(Bill, C100), do(drop(Bill, C100), do(register(Bill, C100), S_0)))]$,

which is

$$\begin{aligned}&\mathcal{R}^0[(\forall p).prerequ(p, C100) \supset (\exists g).grade(Bill, p, g, S_0) \wedge g \geq 50] \wedge\\ &\mathcal{R}^1[enrolled(Bill, C100, do(register(Bill, C100), S_0))] \wedge\\ &\mathcal{R}^2[enrolled(Bill, C100, do(drop(Bill, C100), do(register(Bill, C100), S_0)))].\end{aligned}$$

This yields

$$\begin{aligned}&\{(\forall p).prerequ(p, C100) \supset (\exists g).grade(Bill, p, g, S_0) \wedge g \geq 50\} \wedge\\ &true \wedge\\ &false\end{aligned}$$

so the transaction sequence is indeed illegal.

Consider next the sequence

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

We first compute

$$\begin{aligned}&\mathcal{R}^0[precond(change(Bill, C100, 60), S_0)] \wedge\\ &\mathcal{R}[precond(register(Sue, C200), do(change(Bill, C100, 60), S_0))] \wedge\\ &\mathcal{R}^2[precond(drop(Bill, C100), do(register(Sue, C200),\\ &\qquad\qquad do(change(Bill, C100, 60), S_0)))],\end{aligned}$$

which is

$$\begin{aligned}&\mathcal{R}^0[(\exists g')grade(Bill, C100, g', S_0) \wedge g' \neq 60] \wedge\\ &\mathcal{R}^1[(\forall p)prerequ(p, C200) \supset (\exists g)grade(Sue, p, g, do(change(Bill, C100, 60), S_0)) \wedge\\ &\quad g \geq 50] \wedge\\ &\mathcal{R}^2[enrolled(Bill, C100, do(register(Sue, C200), do(change(Bill, C100, 60), S_0)))].\end{aligned}$$

This simplifies to

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

So the transaction sequence is legal iff this formula is entailed by the initial database.

5.2. *Query Evaluation*

We now consider the evaluation of queries in a database state resulting from a given sequence of update transactions. Specifically, we address the following problem: Given a sequence $\tau_1, \ldots, \tau_n$ of ground terms of sort *transaction*, 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

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

Our principal result is the following:

Theorem 5.2 Reiter [43]. 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 transaction. Then if $\tau_1, \ldots, \tau_n$ is a legal transaction sequence,

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

iff

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

Notice that, as in the case of verifying legality, query evaluation reduces to first order theorem proving *in the initial database $\mathcal{D}_{S_0}$*, together with unique names axioms for transactions. Once again, the second order induction axiom is not required.

Corollary 5.1 Relative Consistency. $\mathcal{D}$ is satisfiable iff $\mathcal{D}_{unt} \cup \mathcal{D}_{S_0}$ is.

PROOF. Take $Q(s) = false$ in Theorem 5.2. □

Corollary 5.1 provides an important relative consistency result. It guarantees that we cannot introduce an inconsistency to a "base" theory $\mathcal{D}_{unt} \cup \mathcal{D}_{S_0}$ by augmenting it with the axioms for $<$ and induction, together with successor state and transaction precondition axioms and unique names axioms for states.

The legality condition in Theorem 5.2 is necessary, as the following example shows:

Example Suppose $\mathcal{L}$ has just a 0-ary function symbol T of sort *transaction* and a fluent F. Consider the successor state axiom

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

and the transaction precondition axiom

$$Poss(T, s) \equiv false.$$

Then if $\mathcal{D}_{S_0} = \{F(S_0)\}$,

$$\mathcal{D}_{S_0} \models \mathcal{R}[F(do(T, S_0))],$$

but

$$\mathcal{D} \not\models F(do(T, S_0)).$$

Example: Query Evaluation Consider again the transaction sequence

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

Suppose the query is

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

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

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

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

6. PROVING PROPERTIES OF DATABASE STATES

As indicated in Section 5.1, there is a close analogy between our approach to database updates and the theory of the natural numbers; simply identify S_0 with the natural number 0, and $do(Add1, s)$ with the successor of the natural number s. In effect, a database is a theory in which each "natural number" s has arbitrarily many successors.[6] Just as mathematical induction is necessary to prove anything

[6] There could even be infinitely many successors whenever a transaction function is parameterized by a real number, as for example $change\text{-}salary(p, \$)$.

interesting about the natural numbers, so also is induction required to prove general properties of database states. This section is devoted to formulating some induction principles suitable for this task, and to providing an account of integrity constraints in this setting. As we shall see, integrity constraints will emerge as inductively derivable general properties of database states.

Let W be a unary predicate variable of $\mathcal{L}$. Using the axioms (6), (5.1), (5.2) and (5.3), Reiter [42] derives the following second order induction principle, suitable for proving properties of states s when $S_0 \leq s$:

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

Frequently, we shall want to prove sentences of the form

$$(\forall s,s').S_0 \leq s \wedge s \leq s' \supset T(s,s').$$

Towards that end, Reiter [42] derives the the following induction principle, suitable for proving properties of pairs of states s and s' when $S_0 \leq s \wedge s \leq s'$:

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

6.1. Induction and the Verification of Integrity Constraints

In the theory of databases, an *integrity constraint* specifies what counts as a legal database state; it is a property that every database state must satisfy. The concept of an integrity constraint is intimately connected with that of database *evolution*; no matter how the database evolves, the constraint must be true in all database futures. Accordingly, it is natural to represent these as sentences, universally quantified over states. For example, no one may have two different grades for the same course in any database state:

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

In a personnel database, we might require that salaries must never decrease during the evolution of the database:

$$(\forall s,s')(\forall p,\$,\$').S_0 \leq s \wedge s \leq s' \wedge sal(p,\$,s) \wedge sal(p,\$',s') \supset \$ \leq \$'.$$[7]

The intuition that constraints are sentences which must be true in all database states leads to the following:

Definition: Constraint Satisfaction A database *DB satisfies* an integrity constraint *IC* iff the database entails the constraint:

$$DB \models IC.$$[8]

[7] The symbol $\leq$ in $\$ \leq \$'$ is the usual ordering relation on the reals, and is not to be confused with our ordering relation on states.

[8] This definition should be contrasted with those in Reiter [38, 41]. It seems that there is not a unitary concept of integrity constraint in database theory, and that there are many subtleties

Notice the assumption underlying the above notion of an integrity constraint and its satisfaction by a database: Constraints are sentences quantified over states, and in the situation calculus, states change *only by virtue of transaction "occurrences"*. So when we speak of a constraint being true in all database states, we mean that arbitrary transaction sequences preserve the truth of the constraint. In other words, we are here imagining that *the only way a database evolves is through transactions.* Consider a database which initially has no information about John's marital status. After several transactions, we discover that he is married. If there is a transaction for marriage events, and if John's marriage in the real world is the next event to be recorded in the database, then simply add this marriage transaction to the current sequence of transactions, and we are done. On the other hand, if there is no database transaction for marriage events, or if we don't know when he married, then the best we can do is add an assertion to the database that John is married. This change to the database is not the result of a transaction, and therefore cannot be formalized within our transaction-centred approach to database evolution. Our concept of an integrity constraint and its satisfaction would not apply in this setting.[9]There is obviously an intimate connection between this observation and that of Katsuno and Mendelzon ([20]) who argue that there is a difference between updating a database and *revising* it. To formally capture this distinction, they propose a set of *update postulates* which differ from, but are in the same style as, the AGM postulates for revision (Alchourrón, Gärdenfors and Makinson [3]). With respect to the above example, recording a marriage transaction corresponds to an update, while simply recording the fact that John is married corresponds to a revision. For a further discussion of this distinction, see Section 7. In the remainder of this paper, our perspective on integrity constraints and their satisfaction will be exclusively transaction-centred; we do not consider databases evolving under revision operations.

We return now to the problem of verifying constraints. Since this requires showing that some sentence is true in all database states, it is not surprising that induction is required. The following result will provide a useful corollary for verifying integrity constraints by induction.

Lemma 6.1 Reiter [43]. Suppose $G(\vec{x}, s) \in \mathcal{L}$ where $G(\vec{x}, s)$ is simple, s is a state variable, and the free variables of G are among $\vec{x}, s$. Then

$$\mathcal{D}_{ss} \models (\forall a, s).Poss(a, s) \supset (\forall \vec{x}).\{G(\vec{x}, do(a, s)) \equiv \mathcal{R}[G(\vec{x}, do(a, s))]\}.$$

Notation: $IP_{S_0 \leq s \leq s'}(H)$, $IP_{S_0 \leq s}(G)$. When $H(s, s') \in \mathcal{L}$ is a formula with two free variables s and s' of sort *state*, $IP_{S_0 \leq s \leq s'}(H)$ denotes the substitution instances of H for R in the induction principle $(IP_{S_0 \leq s \wedge \leq s'})$. When $G(s) \in \mathcal{L}$ is a formula with one free variable of sort *state*, $IP_{S_0 \leq s}(G)$ denotes the substitution instances of G for W in the induction principle $(IP_{S_0 \leq s})$.

The following is an immediate consequence of Lemma 6.1. We will find it useful for verifying integrity constraints by induction.

involved.

[9] I am grateful to one of the referees for this example, and for pointing out that our approach to integrity constraints applies only to databases whose evolution is governed exclusively by transactions.

Corollary 6.1. Suppose $G(s) \in \mathcal{L}$ and $H(s,s') \in \mathcal{L}$ are both simple, that the state variable s is the only free variable of G, and that the state variables s and s' are the only free variables of H. Then,

$$\begin{aligned}\mathcal{D}_{ss} \models IP_{S_0 \leq s}(G) \equiv & \\ & G(S_0) \wedge \\ & \{(\forall a,s).Poss(a,s) \wedge S_0 \leq s \wedge G(s) \supset \mathcal{R}[G(do(a,s))]\} \\ & \qquad \supset (\forall s).S_0 \leq s \supset G(s),\end{aligned}$$

$$\begin{aligned}&\mathcal{D}_{ss} \models \\ &\quad IP_{S_0 \leq s \leq s'}(H) \equiv \\ &\quad\quad H(S_0,S_0) \wedge \\ &\quad\quad \{(\forall a,s,s').Poss(a,s') \wedge S_0 \leq s \wedge s \leq s' \wedge H(s,s') \supset \mathcal{R}[H(s,do(a,s'))]\} \wedge \\ &\quad\quad \{(\forall a,s).Poss(a,s) \wedge S_0 \leq s \wedge H(s,s) \supset \mathcal{R}[H(do(a,s),do(a,s))]\} \\ &\quad\quad\qquad \supset (\forall s,s').S_0 \leq s \wedge s \leq s' \supset H(s,s').\end{aligned}$$

6.2. Examples of Constraints and Their Verification

Proving a Functional Dependency Consider again the example education database, and the successor state axiom

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

Normally, the relation $grade(st,c,g,s)$ is functional in its third argument. Such functional dependencies are examples of so-called *static* integrity constraints. Suppose $\mathcal{D}_{S_0}$ contains the initial functional dependency

$$(\forall st,c,g,g').grade(st,c,g,S_0) \wedge grade(st,c,g',S_0) \supset g = g'. \tag{6.1}$$

We prove that transaction sequences preserve this functional dependency, namely that

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

This we do by invoking the first result of Corollary 6.1 with

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

Therefore, we must prove the following two sentences:
$G(S_0)$, which is the initial functional dependency (6.1).

$$\begin{aligned}&(\forall a,s).Poss(a,s) \wedge S_0 \leq s \wedge \\ &\quad \{(\forall st,c,g,g').grade(st,c,g,s) \wedge grade(st,c,g',s) \supset g = g'\} \supset \\ &\quad\quad [(\forall st,c,g,g')\{a = change(st,c,g) \vee \\ &\qquad\qquad\qquad grade(st,c,g,s) \wedge (\forall g'')a \neq change(st,c,g'')\} \wedge \\ &\qquad\quad \{a = change(st,c,g') \vee \\ &\qquad\qquad\qquad grade(st,c,g',s) \wedge (\forall g'')a \neq change(st,c,g'')\} \\ &\qquad\qquad \supset g = g'].\end{aligned}$$

This has an easy proof using the unique names axioms for transactions. Notice that the proof does not appeal to any transaction precondition axioms.

Proving a Dynamic Integrity Constraint The classic example of a dynamic integrity constraint is that a person's salary must not decrease:

$$(\forall s, s', p, \$, \$').S_0 \leq s \wedge s \leq s' \supset sal(p, \$, s) \wedge sal(p, \$', s') \supset \$ \leq \$'. \tag{6.2}$$

We shall require a transaction precondition axiom stating that the prerequisite for changing a person's salary is that the new salary be greater than the old:

$$Poss(change\text{-}sal(p, \$), s) \equiv (\exists \$').sal(p, \$', s) \wedge \$' < \$. \tag{6.3}$$

Initially, the relation *sal* is functional in its second argument:

$$(\forall p, \$, \$').sal(p, \$, S_0) \wedge sal(p, \$', S_0) \supset \$ = \$'. \tag{6.4}$$

Finally, we assume the following successor state axiom for *sal*:

$$\begin{aligned} Poss(a, s) \supset \{&sal(p, \$, do(a, s)) \equiv a = change\text{-}sal(p, \$) \vee \\ &sal(p, \$, s) \wedge a \neq fire(p) \wedge (\forall \$')a \neq change\text{-}sal(p, \$')\}. \end{aligned}$$

Now, to prove (6.2) we appeal to the second result of Corollary 6.1 with

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

Accordingly, we must prove the following three sentences:

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

   This follows from the initial functional dependency axiom (6.4).

2. $$\begin{aligned} &(\forall a, s, s').Poss(a, s') \wedge S_0 \leq s \wedge s \leq s' \wedge \\ &\qquad [(\forall p, \$, \$').sal(p, \$, s) \wedge sal(p, \$', s') \supset \$ \leq \$'] \\ &\supset \\ &\qquad [(\forall p, \$, \$').sal(p, \$, s) \wedge \{a = change\text{-}sal(p, \$') \vee \\ &\qquad\qquad sal(p, \$', s') \wedge a \neq fire(p) \wedge (\forall \$'')a \neq change\text{-}sal(p, \$'')\} \\ &\qquad\quad \supset \$ \leq \$']. \end{aligned}$$

   The straightforward proof requires the transaction precondition axiom (6.3).

3. $$\begin{aligned} &(\forall a, s).Poss(a, s) \wedge S_0 \leq s \wedge \{(\forall p, \$, \$').sal(p, \$, s) \wedge sal(p, \$', s) \supset \$ \leq \$'\} \\ &\supset \\ &\qquad (\forall p, \$, \$').\{a = change\text{-}sal(p, \$) \vee \\ &\qquad\qquad sal(p, \$, s) \wedge a \neq fire(p) \wedge (\forall \$'')a \neq change\text{-}sal(p, \$'')\} \wedge \\ &\qquad \{a = change\text{-}sal(p, \$') \vee \\ &\qquad\qquad sal(p, \$', s) \wedge a \neq fire(p) \wedge (\forall \$'')a \neq change\text{-}sal(p, \$'')\} \\ &\qquad\quad \supset \$ \leq \$'. \end{aligned}$$

   This has a simple proof using unique names axioms for transactions.

An Example of Casanova and Furtado [8]

Suppose no one who has been fired can ever be rehired:

$$Poss(hire(p), s) \equiv \neg trans(fire(p), s) \wedge \neg emp(p, s). \tag{6.5}$$

Intuitively, $trans(a,s)$ means that the transaction a is part of the transaction sequence leading from S_0 to s. Formally, we have the successor state axiom

$$Poss(a,s) \supset \{trans(a', do(a,s)) \equiv a = a' \vee trans(a', s)\}, \tag{6.6}$$

together with the initial state axiom

$$\neg trans(a, S_0). \tag{6.7}$$

Finally, assume the following successor state axiom for the relation emp:

$$Poss(a,s) \supset \{emp(p, do(a,s)) \equiv a = hire(p) \vee emp(p,s) \wedge a \neq fire(p)\}.$$

We wish to prove that any employed person who subsequently becomes unemployed will forever thereafter remain unemployed:

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

This is easy to prove using transitivity of $\leq$ (a fact which is easily proved by induction) together with the following three sentences:

$$(\forall a, s, s').S_0 \leq s \wedge s \leq s' \wedge trans(a,s) \supset trans(a,s'). \tag{6.8}$$

$$(\forall p, s).S_0 \leq s \wedge trans(fire(p), s) \supset \neg emp(p,s). \tag{6.9}$$

$$(\forall p, s, s').S_0 \leq s \wedge s \leq s' \wedge emp(p,s) \wedge \neg emp(p,s') \supset trans(fire(p), s'). \tag{6.10}$$

Accordingly, we indicate how to prove these.

- Proof of (6.8): Use the second result of Corollary 6.1, and (6.7).
- Proof of (6.9): Use the first result of Corollary 6.1, (6.7), unique names axioms for transactions, and the transaction precondition axiom (6.5).
- Proof of (6.10): Use the second result of Corollary 6.1.

7. EXTENSIONS OF THIS APPROACH

We have described a fairly general approach to specifying update transactions for databases. Nevertheless, within this framework, there remain a number of outstanding problems to be addressed. In order to demonstrate the generality of our situation calculus-based approach to describing evolving databases, we describe some of these research problems, and sketch possible solutions to them within our framework. The proposed solutions are presented with varying degrees of detail, and should be viewed as suggestions for approaching a variety of what, at the moment, are open research problems.

7.1. Transaction Logs and Historical Queries

Using the relation $<$ on states, as defined in Section 5, 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:

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

Was John ever simultaneously enrolled in both $C100$ and $M100$?

$$(\exists s).S_0 \leq s \wedge s \leq do(\mathbf{T}, S_0) \wedge enrolled(John, C100, s) \wedge \\ enrolled(John, M100, s). \tag{7.1}$$

Has Sue always worked in department 13?

$$(\forall s).S_0 \leq s \wedge s \leq do(\mathbf{T}, S_0) \supset emp(Sue, 13, s). \tag{7.2}$$

The rest of this section sketches an approach to answering historical queries of this kind. The approach is of interest because it reduces the evaluation of such queries to evaluations in the initial database state, together with conventional list processing techniques on the transaction *log* consisting of the list of those transactions which are assumed to have taken place.

Begin by defining an abbreviation, *occurs-between*(a, s, s') whose intended interpretation is that situation s' is accessible from situation s via some sequence of executable transactions, and that transaction a is one of the transactions in this sequence:

$$occurs\text{-}between(a, s, s') \stackrel{\Delta}{=} (\exists s'').s < do(a, s'') \leq s'.$$

If we think of a state as a list of all the transactions leading from S_0 to that state, then provided state s' is legal (see Section 5.1), *occurs-between*(a, s, s') is true iff a is a member of the "list difference" of s' and s, where state s is a "sublist" of s'. For example, if

$$do([register(John, C100), drop(Bill, C100), drop(Mary, C100), \\ drop(John, M100)], S_0)$$

is legal, then

$$occurs\text{-}between(drop(Mary, C100), \\ do([register(John, C100)], S_0), \\ do([register(John, C100), drop(Bill, C100), \\ drop(Mary, C100), drop(John, M100)], S_0)),$$

is true, whereas

$$occurs\text{-}between(register(Mary, C100), \\ do([register(John, C100)], S_0), \\ do([register(John, C100), drop(Bill, C100), \\ drop(Mary, C100), drop(John, M100)], S_0)),$$

is false (assuming unique names axioms for transactions).

Example 1:

We begin by showing how to answer query (7.2). Towards that end, we first derive a suitable closed-form solution for fluent F. Assume that F's successor state axiom has the following syntactic form:

$$Poss(a, s) \supset [F(\vec{x}, do(a, s)) \equiv \gamma_F^+(\vec{x}, a, s) \vee F(\vec{x}, s) \wedge \neg\gamma_F^-(\vec{x}, a, s)]. \tag{7.3}$$

Here $\gamma_F^+(\vec{x}, a, s)$ and $\gamma_F^-(\vec{x}, a, s)$ are arbitrary first order formulas with free variables among $\vec{x}, a$ and s. All of the successor state axioms used in the examples of this paper have this syntactic form. Using this and the induction principle $(IP_{S_0 \leq s})$, it is possible to prove:

$$(\forall s').S_0 \leq s' \supset \{[(\forall s)(S_0 \leq s \leq s' \supset F(\vec{x}, s))] \equiv \\ F(\vec{x}, S_0) \land \neg(\exists a, s'').do(a, s'') \leq s' \land \gamma_F^-(\vec{x}, a, s'')\}.$$

Suppose that $\gamma_F^-(\vec{x}, a, s)$ is independent of s, i.e. it nowhere mentions a state variable s. To indicate this, we write it as $\gamma_F^-(\vec{x}, a)$. Then, using the above sentence for F and the abbreviation for *occurs-between*, it is easy to prove that:

$$(\forall s').S_0 \leq s' \supset \{[(\forall s)(S_0 \leq s \leq s' \supset F(\vec{x}, s))] \equiv \\ F(\vec{x}, S_0) \land \neg(\exists a) occurs\text{-}between(a, S_0, s') \land \gamma_F^-(\vec{x}, a)\}. \quad (7.4)$$

Suppose, for the sake of the example, that the successor state axiom for *emp* is:

$$Poss(a, s) \supset emp(p, d, do(a, s)) \equiv \\ a = hire(p, d) \lor emp(p, d, s) \land a \neq fire(p) \land a \neq quit(p).$$

Using this successor state axiom, it is easy to show that the following follows from (7.4):

$$(\forall s').S_0 \leq s' \supset \{[(\forall s)(S_0 \leq s \leq s' \supset emp(p, d, s))] \equiv \\ emp(p, d, S_0) \land \neg occurs\text{-}between(fire(p), S_0, s') \land \neg occurs\text{-}between(quit(p), S_0, s')\}.$$

Using this, together with the assumption that the transaction sequence $\mathbf{T}$ is legal, we get that the original query is equivalent to:

$$emp(Sue, 13, S_0) \land \\ \neg occurs\text{-}between(fire(Sue), S_0, do(\mathbf{T}, S_0)) \land \\ \neg occurs\text{-}between(quit(Sue), S_0, do(\mathbf{T}, S_0)).$$

This form of the original query reduces query evaluation to evaluation in the initial database state, together with simple *list processing* on the database log $\mathbf{T}$ of those transactions leading to the current database state. We can verify that *Sue* has always been employed in department 13 in the following way:

1. Verify that she was initially employed in department 13, and,
2. Show that neither $fire(Sue)$ nor $quit(Sue)$ are members of list $\mathbf{T}$.[10]

Example 2:

We now consider evaluating the first query (7.1) in the same list processing spirit. First, we introduce a new abbreviation $last(a, s)$ meaning that a is the last transaction of the sequence s:

$$last(s, a) \triangleq (\exists s')s = do(a, s').$$

For example,

$$last(do([drop(Mary, C100), register(John, C100)], S_0), register(John, C100))$$

[10] The correctness of this simple-minded list processing procedure relies on some assumptions, notably, suitable unique names axioms.

is true, while

$$last(do([drop(Mary, C100), drop(John, C100)], S_0), register(John, C100))$$

is false, assuming unique names axioms for transactions.

Next, using (7.3) and the induction principle $(IP_{S_0 \leq s})$, we can derive the following closed-form solution for the fluent F:

$$\begin{array}{l} S_0 \leq s \supset \{F(\vec{x}, s) \equiv \\ \qquad F(\vec{x}, S_0) \wedge \neg(\exists a, s')[do(a, s') \leq s \wedge \gamma_F^-(\vec{x}, a, s')] \vee \\ \qquad (\exists a', s')[do(a', s') \leq s \wedge \gamma_F^+(\vec{x}, a', s') \wedge \\ \qquad\qquad \neg(\exists a'', s'')[do(a', s') < do(a'', s'') \leq s \wedge \gamma_F^-(\vec{x}, a'', s'')]]\}. \end{array}$$

Suppose that $\gamma_F^-(\vec{x}, a, s)$ and $\gamma_F^+(\vec{x}, a, s)$ are both independent of s, i.e. they nowhere mention a state variable s. To indicate this, we write them as $\gamma_F^-(\vec{x}, a)$ and $\gamma_F^+(\vec{x}, a)$, respectively. Then, using the above closed form solution for F and the abbreviations for *last* and *occurs-between*, it is easy to prove that:

$$\begin{array}{l} S_0 \leq s \supset \{F(\vec{x}, s) \equiv \\ \qquad F(\vec{x}, S_0) \wedge \neg(\exists a)[\mathit{occurs\text{-}between}(a, S_0, s) \wedge \gamma_F^-(\vec{x}, a)] \vee \\ \qquad (\exists a', s')[last(s', a') \wedge s' \leq s \wedge \gamma_F^+(\vec{x}, a') \wedge \\ \qquad\qquad \neg(\exists a'')[\mathit{occurs\text{-}between}(a'', s', s) \wedge \gamma_F^-(\vec{x}, a'')]]\}. \end{array} \tag{7.5}$$

Suppose the successor state axiom for *enrolled* is:

$$\begin{array}{l} Poss(a, s) \supset \{enrolled(st, c, do(a, s)) \equiv \\ \qquad\qquad a = register(st, c) \vee enrolled(st, c, s) \wedge a \neq drop(st, c)\}. \end{array}$$

Using this successor state axiom, it is easy to show that the following closed form solution for *enrolled* follows from (7.5):

$$\begin{array}{l} S_0 \leq s \supset [enrolled(st, c, s)) \equiv \\ \qquad \{enrolled(st, c, S_0) \wedge \neg \mathit{occurs\text{-}between}(drop(st, c), S_0, s) \vee \\ \qquad (\exists s').s' \leq s \wedge last(s', register(st, c)) \wedge \\ \qquad\qquad \neg \mathit{occurs\text{-}between}(drop(st, c), s', s)\}]. \end{array} \tag{7.6}$$

Then, on the assumption that the transaction sequence $\mathbf{T}$ is legal, it is simple to prove that the query (7.1) is equivalent to:

$$
\begin{array}{l}
\left\{\begin{array}{l} enrolled(John, C100, S_0) \wedge \\ enrolled(John, M100, S_0) \end{array}\right\} \\
\vee \\
\left\{\begin{array}{l} enrolled(John, C100, S_0) \wedge \\ (\exists s).S_0 \leq s \leq do(\mathbf{T}, S_0) \wedge \\ \neg\, occurs\text{-}between(drop(John, C100), S_0, s) \wedge \\ last(s, register(John, M100)) \end{array}\right\} \\
\vee \\
\left\{\begin{array}{l} enrolled(John, M100, S_0) \wedge \\ (\exists s).S_0 \leq s \leq do(\mathbf{T}, S_0) \wedge \\ \neg\, occurs\text{-}between(drop(John, M100), S_0, s) \wedge \\ last(s, register(John, C100)) \end{array}\right\} \\
\vee \\
\left\{\begin{array}{l} (\exists s, s').S_0 < s' < s \leq do(\mathbf{T}, S_0) \wedge \\ last(s', register(John, M100)) \wedge \\ last(s, register(John, C100)) \wedge \\ \neg\, occurs\text{-}between(drop(John, M100), s', s) \end{array}\right\} \\
\vee \\
\left\{\begin{array}{l} (\exists s, s').S_0 < s' < s \leq do(\mathbf{T}, S_0) \wedge \\ last(s', register(John, C100)) \wedge \\ last(s, register(John, M100)) \wedge \\ \neg\, occurs\text{-}between(drop(John, C100), s', s) \end{array}\right\}
\end{array}
$$

Despite its apparent complexity, this sentence also has a simple list processing reading; we can verify that *John* is simultaneously enrolled in $C100$ and $M100$ in some previous database state provided one of the following conditions holds:

1. *John* was initially enrolled in both $C100$ and $M100$.
2. *John* was initially enrolled in $C100$. Moreover, $\mathbf{T}$ has a sublist (loosely denoted by s) whose last element is $register(John, M100)$ and which does not contain $drop(John, C100)$.
3. *John* was initially enrolled in $M100$. Moreover, $\mathbf{T}$ has a sublist s whose last element is $register(John, C100)$ and which does not contain $drop(John, M100)$.
4. $\mathbf{T}$ has a sublist s, which in turn has a sublist s', s' ends with $register(John, M100)$, s ends with $register(John, C100)$, and $drop(John, M100)$ is not a member of the list difference of s and s'.
5. $\mathbf{T}$ has a sublist s, which in turn has a sublist s', s' ends with $register(John, C100)$, s ends with $register(John, M100)$, and $drop(John, C100)$ is not a member of the list difference of s and s'.

Historical queries need not reference only the past; meaningful queries can be posed about the future, for example, given the current database state (which we shall take to be S_0) is it possible for *John* ever to graduate?

$$(\exists s).S_0 \leq s \wedge graduate(John, s).$$

Answering queries of this form is precisely the problem of plan synthesis in AI (Green [15]). Moreover, from a constructive proof of such a query, one can obtain a sequence of transactions leading to a state in which the query is true. This means that in the event that the query's answer is "yes", one can also provide a sequence of steps which, if executed, is guaranteed to lead to the desired state. Thus, for the example at hand, one would be able to compute answers of the form "Yes, it is possible for *John* to graduate, provided he registers for $C400$ and obtains a passing grade for it." For the class of databases of this paper, Reiter [43] shows how regression provides a sound and complete evaluator for such queries.

7.2. Complexity of Query Evaluation

The results of the previous section on transaction logs and historical queries provide a basis for a complexity analysis of query evaluation. As an indication of how such an analysis might proceed, consider the problem of evaluating a query in a database state resulting from a given legal sequence $\mathbf{T}$ of transactions, as in Section 5.2. For simplicity, suppose the query is ground and atomic, say, pursuing example 2 of the previous section, $enrolled(John, C100, do(\mathbf{T}, S_0))$. It is easy to see that the following is a consequence of (7.6) and the assumption that $\mathbf{T}$ is legal:

$$\begin{aligned}
&enrolled(st, c, do(\mathbf{T}, S_0)) \equiv \\
&\quad [enrolled(st, c, S_0) \wedge \neg occurs\text{-}between(drop(st, c), S_0, do(\mathbf{T}, S_0)) \vee \\
&\quad (\exists s').s' \leq do(\mathbf{T}, S_0) \wedge last(s', register(st, c)) \wedge \\
&\qquad \neg occurs\text{-}between(drop(st, c), s', do(\mathbf{T}, S_0))].
\end{aligned}$$

Therefore, the query $enrolled(John, C100, do(\mathbf{T}, S_0))$ is logically equivalent to:

$$\begin{aligned}
&enrolled(John, C100, S_0) \wedge \neg occurs\text{-}between(drop(John, C100), S_0, do(\mathbf{T}, S_0)) \vee \\
&(\exists s').s' \leq do(\mathbf{T}, S_0) \wedge last(s', register(John, C100)) \wedge \\
&\quad \neg occurs\text{-}between(drop(John, C100), s', do(\mathbf{T}, S_0)).
\end{aligned}$$

As before, this has a simple list processing reading: *John* is enrolled in $C100$ iff

1. *John* was initially enrolled in $C100$ and $drop(John, C100)$ is not a member of the transaction log $\mathbf{T}$, or
2. $register(John, C100)$ is a member of the log $\mathbf{T}$ and $drop(John, C100)$ does not occur later than it in the log $\mathbf{T}$.

Clearly, the complexity of this procedure is linear in the length of the log $\mathbf{T}$, plus whatever the complexity is of query evaluation in the initial state. Moreover, there is nothing very special about this example, which is to say that under fairly general conditions:[11]

For queries which are ground literals, the complexity of query evaluation using a transaction log adds complexity linear in the length of the log to the complexity of query evaluation in the initial database.

When the initial database is complete, as would be the case when it is relational, a ground query may be evaluated by first computing its atomic subqueries, as indicated above, then combining those answers in the obvious way according to

[11] Specifically, these general conditions are that the successor state axiom be what Lin and Reiter [28] call *context free*.

the sentential structure of the original query. This provides a tolerable algorithmic complexity for query evaluation. When $\mathcal{D}_{S_0}$ is incomplete, then we do not have this query decompositional structure, and it appears that we must resort to the full generality of regression, as in Section 5.2. As it happens in this case, a complexity analysis in the length of the log is still possible; moreover, the complexity turns out to be tolerable for successor state axioms having a suitable syntactic form. Since these considerations take us too far from the main thrust of this paper, we do not pursue these ideas any further here, except to observe that a rich complexity theory for transaction processing appears to be possible within the framework of the situation calculus.

7.3. Actualizing Transactions

Recall that within our approach to specifying transactions, all updates are virtual; the database is never physically changed. Instead, the axiomatization characterizes all possible database futures under all possible transaction sequences. Determining whether a given formula $Q(s)$ is true in that database state resulting from the transaction log $\mathbf{T}$ reduces to the question of whether the database entails $Q(do(\mathbf{T}, S_0))$ (Section 5.2).

Transaction-intensive databases can lead to extremely long transaction logs, so that regression-based query evaluation (Section 5), or the improved methods of Section 7.1, can become computationally unfeasible, even when the database successor state axioms support linear complexity (in the length of the log) for atomic query evaluation. In such cases, it may be profitable to view a transaction as a mapping from one static database to another, in the style of Abiteboul [1]. From this perspective, a database transaction can be implemented as a physical modification of the current database to yield the updated database which *actualizes* the transaction. In the case of relational databases, such transactions are normally actualized by suitable insertions/deletions of tuples into/from the relational tables of the database. Generally speaking, such static databases suppress all references to the state argument of their corresponding situation calculus specification; they are meant to represent those sentences which would be true in that situation calculus state corresponding to the suppressed state argument.

This idea that transactions are mappings from static databases to static databases is intuitively very appealing; indeed, it informs many approaches to database updates in the literature (e.g. Abiteboul [1], Bonner and Kifer [6], Fagin, Ullman and Vardi [10], Ginsberg and Smith [12], Guessoum and Lloyd [16, 17], Kakas and Mancarella [19]). Surprisingly, this idea is not as simple as it appears on the surface. Lin and Reiter [28] show that even when the initial database is first order (i.e. represents a finite set of first order situation calculus sentences whose only state argument is S_0), the successor database which actualizes the transaction need not be first order definable. It is, however, always second order definable.

This negative result leads to the natural question: when does a situation calculus specification admit a realization in terms of transaction mappings from first order static databases to first order static databases? Reiter and Lin [28, 27] provide two conditions under which this is possible, together with a systematic procedure for computing the successor database from the initial one:

1. When the database is relational.

2. When the database consists of ground literals (but need not be complete), and the successor state axioms have a certain general syntactic form.

These considerations lead one naturally to address the problem of updates for relational databases with null values of the kind denoting existing but unknown individuals. A first order axiomatization of this setting was provided by Reiter [39]. While we have not worked out the details, it is clear that the ideas of this paper can be combined with those of (Reiter [39]) to provide a logical specification of the correct treatment of null values under updates for relational databases. With such a specification in hand, it should be possible to characterize transaction mappings from static databases to static databases, as discussed above, which are provably correct with respect to this specification.

A final consideration concerns the trade-offs to be expected, in particular database application settings, between the approach emphasized in this paper based on transaction logs, and the more conventional treatment of transactions in database systems that involves actualizing each transaction as it is received. It is difficult to provide a formal comparison between these two approaches; neither is uniformly better than the other. Consider a database log of length n. In the case where query evaluation has complexity n, one might think that for large n it would be more efficient to adopt the transaction actualizing approach. But this requires n calculations of the successor databases, and each of these calculations may be nontrivial, or even impossible in first order logic, when the initial database is not relational. Of course, these n database actualizations will take place over the lifetime of the database, so in many cases, there will be sufficient database idle time over its lifetime to make this conventional approach computationally feasible. On the other hand, if the database application is transaction intensive, with little need for query evaluation, the approach based on transaction logs is more attractive. This is so especially when the database is required to process transactions in real time, and there is not enough idle time to perform the transaction actualizing computations. This is the case for the applications to robotics which we are pursuing in the Cognitive Robotics Project at the University of Toronto. Some of the theoretical and computational foundations for this work are provided by the approach to database logs and query evaluation described in this paper. We have found that an approach based exclusively on actualizing transactions is not feasible in this setting, partly because of real time constraints, partly because in this application, transaction logs may *shrink* as well as expand because rollbacks in the log occur whenever the robot's projected behaviour would lead to a dead-end (or worse), in which case backtracking is necessary to the last point in the log in which an alternative behavioural action was possible. Accordingly, we have opted for a mixed strategy in which a database log is maintained, and the robot's "mental idle time" (corresponding to the time it is performing physical activities) is used for the purpose of actualizing the current log. For a description of this application, and the reasons for some of our design decisions regarding logs vs. actualizing transactions, see Lespérance et al [24].

7.4. Updates in the Logic Programming Context

Our approach to database updates can be implemented in a straightforward way as a logic program, thereby directly complementing the logic programming perspective on databases (Minker [36]). For example, the axiomatization of the education example of Section 3 has the following representation as clauses:

Successor State Axiom Translation:

$$enrolled(st, c, do(register(st, c), s)) \leftarrow Poss(register(st, c), s).$$

$$enrolled(st, c, do(a, s)) \leftarrow a \neq drop(st, c), enrolled(st, c, s), Poss(a, s).$$

$$grade(st, c, g, do(change(st, c, g), s)) \leftarrow Poss(change(st, c, g), s).$$

$$grade(st, c, g, do(a, s)) \leftarrow not\ R(a, st, c), grade(st, c, g, s), Poss(a, s).$$

$$R(change(st, c, g'), st, c).^{12}$$

Transaction Precondition Axiom Translation:

$$Poss(register(st, c), s) \leftarrow not\ P(st, c, s).$$

$$P(st, c, s) \leftarrow prerequ(p, c), not\ Q(st, p, s).$$

$$Q(st, p, s) \leftarrow grade(st, p, g, s), g \geq 50.^{13}$$

$$Poss(change(st, c, g), s) \leftarrow grade(st, c, g', s), g \neq g'.$$

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

With a suitable clausal form for $\mathcal{D}_{S_0}$, it would then be possible to evaluate queries against updated databases, for example

$$\leftarrow enrolled(John, C200, do(register(Mary, C100), do(drop(John, C100), S_0))).$$

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. In this connection, notice that the equivalences in the successor state and transaction precondition axioms are reminiscent of Clark's [9] completion semantics for logic programs, and our unique names axioms for states and transactions provide part of the equality theory required for Clark's semantics (Lloyd [31], pp.79, 109).

[12] We have here invoked some of the program transformation rules of (Lloyd [31], p.113) to convert the non-clausal formula

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

to a Prolog executable form. R is a new predicate symbol.

[13] We have here invoked some of the program transformation rules of (Lloyd [31], p.113) to convert the non-clausal formula

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

to a Prolog executable form. P and Q are new predicate symbols.

7.5. Views

In our setting, a *view* is a fluent $V(\vec{x}, s)$ defined in terms of so-called *base* predicates:

$$(\forall \vec{x}, s).V(\vec{x}, s) \equiv \mathcal{B}(\vec{x}, s), \tag{7.7}$$

where $\mathcal{B}$ is a simple formula with free variables among $\vec{x}$ and s, and which mentions only base predicates.[14]Unfortunately, sentences like (7.7) pose a problem for us because they are precluded by their syntax from the databases considered in this paper. However, we can accommodate nonrecursive views by representing them as follows:

$$(\forall \vec{x}).V(\vec{x}, S_0) \equiv \mathcal{B}(\vec{x}, S_0), \tag{7.8}$$

$$(\forall a, s).Poss(a, s) \supset (\forall \vec{x}).V(\vec{x}, do(a, s)) \equiv \mathcal{R}[\mathcal{B}(\vec{x}, do(a, s))].^{15} \tag{7.9}$$

Sentence (7.8) is a perfectly good candidate for inclusion in $\mathcal{D}_{S_0}$, while (7.9) has the syntactic form of a successor state axiom and hence may be included in $\mathcal{D}_{ss}$.

This representation of views requires some formal justification, which the following theorem provides:

Theorem 7.1. Suppose $V(\vec{x}, s)$ is a fluent of $\mathcal{L}$, and that $\mathcal{B}(\vec{x}, s) \in \mathcal{L}$ is a simple formula which does not mention V and whose free variables are among $\vec{x}, s$. Suppose further that $\mathcal{D}_{ss}$ contains the successor state axiom (7.9) for V, and that $\mathcal{D}_{S_0}$ contains the initial state axiom (7.8). Then,

$$\mathcal{D} \cup \{(IP_{S_0 \leq s})\} \models (\forall s).S_0 \leq s \supset (\forall \vec{x}).V(\vec{x}, s) \equiv \mathcal{B}(\vec{x}, s).$$

Proof. We use result 4 of Corollary 6.1 with $G(s)$ as $(\forall \vec{x}).V(\vec{x}, s) \equiv \mathcal{B}(\vec{x}, s)$. This requires proving the following two antecedent conditions:

1. $G(S_0)$, which is simply the axiom (7.8).
2. The second condition is the formula

$$(\forall a, s).Poss(a, s) \wedge S_0 \leq s \wedge \{(\forall \vec{x}).V(\vec{x}, s) \equiv \mathcal{B}(\vec{x}, s)\} \supset \\ \mathcal{R}[(\forall \vec{x}).V(\vec{x}, do(a, s)) \equiv \mathcal{B}(\vec{x}, do(a, s))].$$

By the properties of the regression operator $\mathcal{R}$, this is the same as

$$(\forall a, s).Poss(a, s) \wedge S_0 \leq s \wedge \{(\forall \vec{x}).V(\vec{x}, s) \equiv \mathcal{B}(\vec{x}, s)\} \supset \\ (\forall \vec{x}).\mathcal{R}[V(\vec{x}, do(a, s))] \equiv \mathcal{R}[\mathcal{B}(\vec{x}, do(a, s))].$$

Using the successor state axiom (7.9) for V, this becomes

$$(\forall a, s).Poss(a, s) \wedge S_0 \leq s \wedge \{(\forall \vec{x}).V(\vec{x}, s) \equiv \mathcal{B}(\vec{x}, s)\} \supset \\ (\forall \vec{x}).\mathcal{R}[\mathcal{B}(\vec{x}, do(a, s))] \equiv \mathcal{R}[\mathcal{B}(\vec{x}, do(a, s))],$$

which is identically true.

□

[14] We do not consider recursive views. Views may also be defined in terms of other, already defined views, but everything eventually "bottoms out" in base predicates, so we only consider this case.

Theorem 7.1 informs us that from the initial state and successor state axioms (7.8) and (7.9) we can inductively derive the view definition

$$(\forall s).S_0 \leq s \supset (\forall \vec{x}).V(\vec{x}, s) \equiv \mathcal{B}(\vec{x}, s).$$

This is not quite the same as the view definition (7.7) with which we began this discussion, but it is close enough. It guarantees that in any database state reachable from the initial state S_0, the view definition (7.7) will be true. We take this as sufficient justification for representing views within our framework by the axioms (7.8) and (7.9).

7.6. State Constraints and the Ramification and Qualification Problems

Recall that our definition of a database (Section 5.1) does not admit state-dependent axioms, except those of $\mathcal{D}_{S_0}$ referring only to the initial state S_0. For example, we are prevented from including in a database a statement requiring that any student enrolled in $C200$ must also be enrolled in $C100$.

$$(\forall s, st).S_0 \leq s \wedge enrolled(st, C200, s) \supset enrolled(st, C100, s). \tag{7.10}$$

In a sense, such a state-dependent constraint should be redundant, since the successor state axioms, because they are equivalences, uniquely determine all future evolutions of the database given the initial database state S_0. The information conveyed in axioms like (7.10) must already be embodied in $\mathcal{D}_{S_0}$ together with the successor state and transaction precondition axioms. We have already seen hints of this observation. In Section 6 we showed how the functional dependency

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

is an inductive entailment of the example education database. Similarly, in Section 6 we argued that dynamic integrity constraints should be viewed as inductive entailments of the database, and we gave several examples of such derivations. Finally, Theorem 7.1 shows that the view definition

$$(\forall s).S_0 \leq s \supset (\forall \vec{x}).V(\vec{x}, s) \equiv \mathcal{B}(\vec{x}, s).$$

is an inductive entailment of the database containing the initial state axiom (7.8) and the successor state axiom (7.9).

These considerations suggest that a *state constraint* can be broadly conceived as any sentence of the form

$$(\forall s_1, \ldots, s_n).s_i \leq s_j \wedge S_0 \leq s_k \wedge \cdots \supset W(s_1, \ldots, s_n), \tag{7.11}$$

and that a database is said to *satisfy* this constraint iff the database inductively entails it.[16]This perspective on state constraints – that they are inductive entailments of the database – provides a unifying view of the classical notions of static and dynamic integrity constraints. In our setting, a static integrity constraint is simply a sentence of the form (7.11) with $n = 1$, i.e., a sentence true in all states s accessible from S_0, while a dynamic constraint relates two or more accessible states.

[16] This definition should be contrasted with that of Reiter [38].

Aside from this syntactic difference, they have the same logical status in our theory, namely as sentences which must be entailed by the database.

The fact that state constraints like (7.10) must be inductive entailments of a database does not of itself dispense with the problem of how to deal with such constraints in defining the database. For in order that a state constraint be an inductive entailment, the successor state axioms must be so chosen as to guarantee this entailment. For example, the original successor state axiom for *enroll* (Section 3) was:

$$\begin{aligned} &Poss(a,s) \supset \{enrolled(st,c,do(a,s)) \equiv \\ &\qquad a = register(st,c) \vee enrolled(st,c,s) \wedge a \neq drop(st,c)\}. \end{aligned} \tag{7.12}$$

As one would expect, this does not inductively entail (7.10). One way to accommodate the state constraint (7.10), is to change this successor state axiom to:

$$\begin{aligned} &Poss(a,s) \supset \{enrolled(st,c,do(a,s)) \equiv \\ &\quad a = register(st,c) \vee c = C100 \wedge a = register(st,C200) \vee \\ &\quad enrolled(st,c,s) \wedge a \neq drop(st,c) \wedge [c = C200 \supset a \neq drop(st,C100)]\}. \end{aligned} \tag{7.13}$$

It is now simple to prove that, provided $\mathcal{D}_{S_0}$ contains the unique names axiom $C100 \neq C200$ and the initial instance of (7.10),

$$enrolled(st,C200,S_0) \supset enrolled(st,C100,S_0), \tag{7.14}$$

then (7.10) is an inductive entailment of the database.

This, however, is not the only way to accommodate the state constraint (7.10). Another is to view (7.10) as implicitly imposing a further constraint on the preconditions of the transaction $register(st,C200)$, namely that st be enrolled in $C100$. Recall that the original transaction precondition axiom for *register* (with reference to the example database of Section 3) was:

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

Now, to accommodate the state constraint (7.10), we can change this axiom to:

$$\begin{aligned} &Poss(register(st,c),s) \equiv \\ &\qquad \{(\forall p)[prerequ(p,c) \supset (\exists g).grade(st,p,g,s) \wedge g \geq 50] \wedge \\ &\qquad [c = C200 \supset enrolled(st,C100,s)]\}. \end{aligned} \tag{7.15}$$

As before, it is simple to prove that, provided $\mathcal{D}_{S_0}$ contains the unique names axiom $C100 \neq C200$ and the initial instance (7.14) of (7.10), then the state constraint (7.10) is an inductive entailment of the database.

The example illustrates the subtleties involved in getting the successor state and/or transaction precondition axioms to reflect the intent of a state constraint. These difficulties are a manifestation of the so-called *ramification* (Finger [11]) and *qualification* (McCarthy [34]) problems in artificial intelligence planning domains. Transactions might have ramifications, or *indirect effects.* For the example at hand, the transaction of registering a student in $C200$ can be viewed as having the direct effect of causing the student to be enrolled in $C200$, and the indirect effect of causing her to be enrolled in $C100$ (if she is not already enrolled in $C100$). The modification (7.13) of (7.12) was designed to capture this reading of the state constraint as an indirect effect. The alternative perspective – that the state constraint provides an implicit constraint on transaction precondition axioms – characterizes the qualification problem. For the current example, this is reflected in our choice of the transaction precondition axiom (7.15).

In our setting, the ramification problem is this: Given a static state constraint like (7.10), how can the indirect effects implicit in the state constraint be embodied in the successor state axioms so as to guarantee that the constraint will be an inductive entailment of the database? The qualification problem is this: Given a static state constraint like (7.10), how can its implicit constraints on transaction preconditions be embodied in the transaction precondition axioms so as to guarantee that the constraint will be an inductive entailment of the database? A variety of circumscriptive proposals for addressing these problems (in conjunction with the frame problem) have been proposed in the artificial intelligence literature, notably by Baker [4], Baker and Ginsberg [5], Ginsberg and Smith [12], Lifschitz [26] and Lin and Shoham [30]. Our formulation of the problem in terms of inductive entailments of the database appears to be new. This perspective on constraints is pursued in (Lin and Reiter [29]), where techniques are presented for "compiling" the information implicit in the state constraints into the successor state and transaction precondition axioms.

8. COMPARISON WITH OTHER APPROACHES TO A THEORY OF UPDATES

Relying as it does on the situation calculus, our approach to specifying update transactions differs substantially from other proposals in the literature. We here present a brief comparison with representatives of what we take to be the principal competing logical perspectives on formalizing database updates. We do not consider procedurally oriented approaches (such as Abiteboul [1]).

Logical Status of Database States

In the situation calculus, states are first class citizens over which one can quantify. Quantification over states in the situation calculus amounts to quantification over *sequences* of transactions. For example, one can assert that, or ask whether, there exists a transaction sequence leading to a database state in which such-and-such property is true. This makes historical queries possible (Section 7.1), and provides for a theory of integrity constraints (Section 6.1). This is impossible or extremely awkward to do *within the logic* for those approaches to updates formalized in modal logics e.g. dynamic logic, Manchanda and Warren [32], temporal logic, Casanova and Furtado [8]), or in "path-based" logics (e.g. Bonner and Kifer), for which there is only an implicit notion of state.

Logical Status of Transactions

A feature of the situation calculus, related to that of states as first class citizens, is that transactions and transaction sequences are first order *terms*, in contrast to the approaches of Manchanda and Warren and of Bonner and Kifer in which transactions are predicates. This precludes talking *about* transactions within the logic (e.g. that all transactions of a certain kind must have such-and-such properties). It also precludes the ability to analyze, within the logic, the features of a given transaction sequence, for example, that a given transaction sequence mentions a *register* transaction. We made extensive use of this property of the situation calculus in our analysis of historical queries (Section 7.1).

Transaction-Centred vs. Update-Centred Theories of Updates

Like the work of Abiteboul and Vianu [2], our approach to a theory of updates is *transaction-centred*, meaning that an update is possible only when the database provides for a suitable prespecified transaction corresponding to the desired update. For our example education database, it is possible to alter a student's grade only because there is a prespecified grade-changing transaction $change(st, c, g)$ in the database together with an axiomatization of the intended effects of this transaction; without such a prespecified transaction, it would be impossible to change a grade in this database. Thus, for transaction-centred databases, updates with arbitrary sentences are not permitted. A wide variety of update proposals in the literature are not transaction-centred; they provide for updates of a database with arbitrary sentences (e.g. the model theoretic approaches of Grahne [13], Katsuno and Mendelzon [20], Grahne, Mendelzon and Revesz ([14]) or Winslett [48], the syntactic approaches of Fagin, Ullman and Vardi [10] or Ginsberg and Smith [12], and the abductive approaches of Guessoum and Lloyd [16, 17] or Kakas and Mancarella [19]). In this respect, such proposals are more general than ours, but this generality comes at a price. With the exception of [14], the model theoretic approaches are based on propositional databases. Grahne, Mendelzon and Revesz ([14]) provide an account for first order models based on the Winslett ordering. Their account assumes that the set of all models of a first order theory is in hand. In this case, they provide a computationally tractable (in the sum of the sizes of these models) update algorithm, but the assumption that these models are in hand, or that their sizes are reasonable, is limiting. Similarly, the syntactic approaches are first order, but provide no systematic update operator. The abductive proposals are first order, but are limited to Prolog deductive databases.

There is also a very important *conceptual* issue related to the distinction between transaction-centred theories of updates, and those which permit updates with arbitrary sentences. Following Keller and Winslett ([21]), Katsuno and Mendelzon observe ([20]) that there is a difference between updating a database and *revising* it. To formally capture this distinction, they propose a set of *update postulates* which differ from, but are in the same style as, the AGM postulates for revision (Alchourrón, Gärdenfors and Makinson [3]). For Keller-Winslett and Katsuno-Mendelzon, updates differ from revisions in that the former result from *event occurrences* which change the state of the world, while the latter result from changes in our theory of what a static world is really like. Notice that, conceptually, this perspective on the nature of updates is transaction-centred; updates occur only in response to events (read "transactions"). On this analysis, the above proposals are really transaction-centred (or, at least, they should be viewed this way), but so far as the database is concerned, these transactions are implicit; they exist in the mind of the user, not of the database. Whenever a user requests an update with a sentence, she has in mind some event of which that sentence is an effect; at least this must be so whenever she is requesting an update, as opposed to a revision.

Now it can be argued that a user may not know the event underlying a proposed update. Consider a user who, observing that the street is now wet, wishes to record this fact in the database. She does not know what event in the world had *street-is-wet* as its effect. Since it is transaction-centred, our approach cannot handle this setting at all (but see below for a proposed approach). But neither, we shall argue, can the Katsuno-Mendelzon theory. Certainly, Katsuno-Mendelzon can, and will, accept this update. One consequence of the resulting updated database will be that if *grass-is-dry* were true of the previous database, it will be true of the updated database. But this is clearly undesirable since the underlying event for *street-is-wet* might have been *rain*, one of whose effects would be $\neg$*grass-is-dry*. (Notice that an equally plausible underlying event might have been *sprinkler-truck*, which would have no effect on the truth value of *grass-is-dry*.) So the Katsuno-Mendelzon theory cannot guarantee an intuitively correct account of updates with a sentence whose underlying event is unknown, the reason being that *all and only* the effects of this event will also be unknown, not only to the database (which knows nothing about events and their effects), but also to the user whose responsibility it is to provide a suitable update sentence consisting of all and only those effects of the event in question. In those cases where all possible underlying events and their effects are known, it may be that the Katsuno-Mendelzon theory can be correctly applied, but with some modification whenever there might be several such events which could explain the observed effect. For example, both *rain* and *sprinkler-truck* explain the observation *street-is-wet*, but neither *grass-is-dry* nor $\neg$*grass-is-dry* should be consequences of the updated (with *street-is-wet*) database whenever the original database entails *grass-is-dry*. One way to achieve this within the Katsuno-Mendelzon framework would be to create *two* databases, one resulting from updating the original database with all the effects of *rain* (including, presumably, both *street-is-wet* and $\neg$*grass-is-dry*), and one resulting from updating the original with all the effects of *sprinkler-truck* (including *street-is-wet*, but not $\neg$*grass-is-dry*). So the picture that emerges is roughly the following: Assume we have in hand a description of all possible events together with their effects. Given an observation which we wish to record in a database, determine all events whose effects include the observation. For each such event, perform the Katsuno-Mendelzon update of the database with all the effects of the event. The resulting *set* of databases represents the update with the original observation.

In general, our conclusion is that before a database can correctly record an update, the database or the user must know what its underlying event is. This is true for all the above update mechanisms, as well as for the transaction-centred approach of this paper. The question remains: Given only the results of some world observation, where the event underlying this observation is unknown, how is an agent (human or database) to perform the update? The answer seems to be: By *inferring* what the underlying event(s) might be. One possible mechanism for this is *abduction* (Poole [37]), which has been applied in a wide variety of settings (diagnosis, natural language, planning) for inferring events which might explain an observation.[17]Combining abduction with a conventional approach to updates would lead to a very rich theory of database evolution, but such considerations take us well beyond the focus of this paper.

We summarize what we take to be the major limitation with all the above approaches. Since updates are responses to events occurring in the world, *it becomes the responsibility of the user, and not the database, to know all the effects on the world of an event, and to request updates which include all and only these effects.* Inadvertantly omitting one such effect, or proposing an inappropriate one, will leave the database in an intuitively incorrect state with respect to the world being

[17] This use of abduction for inferring event occurrences is quite different than the abductive approaches to updates advocated by Guessoum and Lloyd [16, 17] and Kakas and Mancarella [19].

modeled, even though, insofar as the database update mechanism is concerned, everything is fine. The source of the problem is clear: the database has no knowledge of events and their effects. If it did, then a suitable theory of updates would simply provide the user with a way to key in the event she wishes to record, and the database would do the rest. This, then, is the major distinction between our approach and these others. We require *that the database contain knowledge of events and their effects*, whereas these other approaches place the responsibility for knowing, and correctly using this information on the individual issuing the update requests.

Virtual vs. Actualized Updates

Many approaches to a theory of updates (e.g. the model theoretic, syntactic and abductive proposals mentioned above) have in common that an update is a mapping which, for a given database (a logical theory) and sentence, determines another database (another logical theory) which is taken to be the result of the update with the sentence (Section 7.3). This mapping is usually accomplished by the addition/deletion of sentences to/from the current database, yielding a database which *actualizes* the update. In contrast, updates for us are *virtual*; the database itself never changes. We accomplish this by the choice of a suitable ontology in which states are first-class citizens and transactions are first order terms, and by an axiomatization which (implicitly) characterizes all possible future evolutions of the database.

From the perspective of updates as mappings from databases to databases, the very concept of an update is metatheoretic in character, even when the databases themselves are theories in some logic. In other words, the effects of updates are not described *within* the database axiomatization, as they are in our approach, but are defined by mechanisms *external* to the database itself. Any such theory of updates will lack certain desirable properties, for example, the ability to reason, within the database itself, about transaction sequences and integrity constraints (as in Section 6.1), or an object level account of query evaluation for a database that has undergone a sequence of update transactions (as in Section 5). Such capabilities can only be realized metatheoretically in any approach which views updates in terms of addition/deletion of sentences to/from some axiom set.

As Lin and Reiter have shown ([28]), it is not always possible to actualize updates within first order logic, which is to say that there are certain limitations to such an approach to specifying updates.

Primitive vs. Complex Transactions

The ability to define complex transactions in terms of primitive ones is extremely important for a theory of updates. As currently developed, our proposal does not provide a mechanism for defining complex transactions. In contrast, such proposals do exist, notably by Machanda and Warren [32]), based on dynamic logic in the logic programming context, and by Bonner and Kifer [6], based on a new logic specifically tailored to transactions. The latter theory is especially interesting for its rich repertoire of operators for defining new transactions in terms of old. These include sequence, nondeterministic choice, conditionals and iteration. The Bonner-Kifer paper focusses on the definition of complex transactions in terms of *elementary* updates. On the assumption that these elementary updates successfully address the frame problem, any complex update defined in terms of these elementary ones will inherit a correct solution to the frame problem. Unfortunately, Bonner and Kifer do not address the frame problem for these elementary updates; this task is left to the person specifying the database. In this connection, our current proposal can be seen as complementary to that of Bonner and Kifer in that our focus is on addressing the frame problem only for elementary updates, while deferring consideration of this problem for complex transactions. For an extension of the situation calculus to provide for complex transactions along the lines of Bonner and Kifer, see (Levesque, Lin and Reiter [25]).

Classical vs. Other Logics

Unlike proposals based on modal logics, e.g. dynamic logic (Manchanda and Warren [32]) or temporal logic (Casanova and Furtado [8]), or specially tailored logics, e.g. (Bonner and Kifer [6]), ours is based on first order logic (with a second order induction principle). This has the advantage of an established, well understood semantics and proof theory, and it meshes well with the standard perspective of a (static) database as a special kind of first order theory. Moreover, it provides a sound and complete query evaluation mechanism based on goal regression, and an account of database integrity constraints in terms of inductive entailments of the database.

Proving Properties of Database States

In Section 6 we introduced an induction principle suitable for proving properties true in all database states. This feature is particularly important for the purposes of verifying integrity constraints which, from our perspective, are inductive entailments of the database. None of the other logical approaches to a theory of updates with which we are familiar provides for inductive proofs of database states. Indeed, this would appear to be impossible in those approaches which treat transactions as predicates. It is, of course, meaningless for those update theories which are not transaction-centred.

The Event Calculus

The one proposal in the literature closest in spirit to ours is Kowalski's theory of updates based on the event calculus [22]. His axiomatization is first order (with a Prolog semantics), transactions are first order terms (actually, constants), states (in his case, time) are first class citizens, updates are virtual, the approach is transaction-centred, and it addresses the frame problem (using Prolog's negation-as-failure mechanism). Despite these similarities, it is difficult to compare the two approaches, primarily because they appeal to quite different logical foundations. Recently, nevertheless, Kowalski and Sadri [23] compare the situation calculus axioms of this paper with an axiomatization of the event calculus and reveal some interesting relationships between our successor state axioms and their analogue within the event calculus.

9. CONCLUSIONS

The situation calculus is an extremely rich language for the purposes of specifying databases and their evolution under update transactions. In this paper we have presented one way of using the situation calculus for these objectives. Ours is a transaction-centred approach, in which all transactions are treated as primitive.

States are first class citizens, transactions are first order terms and the theory provides an object level account of the effects of updates. We observed that the frame problem is a fundamental obstacle to an adequate formalization of database evolution, and we showed how to axiomatize the effects of elementary transactions in such a way as to overcome this problem. For a certain class of database axiomatizations, incorporating our proposed solution to the frame problem, we gave a sound and complete query evaluation mechanism based on goal regression. We also provided an induction principle, suitable for proving properties of database states under arbitrary sequences of transactions, and for verifying integrity constraints. Finally, we discussed possible extensions of the approach of this paper, including transaction logs and historical queries, the complexity of query evaluation, actualized transactions, logic programming approaches to updates, database views and state constraints.

Acknowledgments
I had a lot of help on this one. Many thanks to Leo Bertossi, Tony Bonner, Alex Borgida, Craig Boutilier, Charles Elkan, Michael Gelfond, Gösta Grahne, Russ Greiner, Joe Halpern, Michael Kifer, Hector Levesque, Vladimir Lifschitz, Fangzhen Lin, Wiktor Marek, John McCarthy, Alberto Mendelzon, John Mylopoulos, Javier Pinto, Len Schubert, Yoav Shoham and Marianne Winslett. The referees' suggestions considerably improved an earlier version of this paper. 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. I am grateful to the Canadian Institute for Advanced Research for granting me a Fellowship providing the release time during which this work was done.

REFERENCES

1. S. Abiteboul. Updates, a new frontier. In *Second International Conference on Database Theory*, pages 1–18. Springer, 1988.
2. S. Abiteboul and V. Vianu. A transaction-based approach to relational database specification. *Journal of the ACM*, 36:759–789, 1989.
3. C.E. Alchourrón, P. Gärdenfors, and D. Makinson. On the logic of theory change: partial meet contraction and revision functions. *Journal of Symbolic Logic*, 50:510–530, 1985.
4. A. Baker. A simple solution to the Yale shooting problem. In R. Brachman, H.J. Levesque, and R. Reiter, editors, *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning (KR'89)*, pages 11–20. Morgan Kaufmann Publishers, Inc., 1989.
5. A. Baker and M. Ginsberg. Temporal projection and explanation. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 906–911, Detroit, MI, 1989.
6. A. Bonner and M. Kifer. Transaction logic programming. Technical report, Department of Computer Science, University of Toronto, 1992.
7. A. Borgida, J. Mylopoulos, and J. Schmidt. The TaxisDL software description language. Technical report, Department of Computer Science, University of Toronto, 1991.
8. M.A. Casanova and A.L. Furtado. A family of temporal languages for the description of transition constraints. In H. Gallaire, J. Minker, and J.M. Nicolas, editors, *Advances in Database Theory*, volume 2, pages 211–238. Plenum Press, New York, 1984.

9. K.L. Clark. Negation as failure. In H. Gallaire and J. Minker, editors, *Logic and Data Bases*, pages 292–322. Plenum Press, New York, 1978.

10. 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.

11. J. Finger. *Exploiting Constraints in Design Synthesis.* PhD thesis, Stanford University, Stanford, CA, 1986.

12. M.L. Ginsberg and D.E. Smith. Reasoning about actions I: A possible worlds approach. *Artificial Intelligence*, 35:165–195, 1988.

13. 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.

14. G. Grahne, A.O. Mendelzon, and P. Revesz. Knowledgebase transformations. In *Proceedings of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 246–260, San Diego, CA, June 2-4, 1992.

15. C. C. Green. Theorem proving by resolution as a basis for question-answering systems. In B. Meltzer and D. Michie, editors, *Machine Intelligence 4*, pages 183–205. American Elsevier, New York, 1969.

16. A. Guessoum and J.W. Lloyd. Updating knowledge bases. *New Generation Computing*, 8(1):71–89, 1990.

17. A. Guessoum and J.W. Lloyd. Updating knowledge bases II. Technical report, University of Bristol, 1991. To appear.

18. 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.

19. A.C. Kakas and P. Mancarella. Database updates through abduction. In *Proceedings VLDB-90*, Brisbane, Australia, 1990.

20. 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.

21. A.M. Keller and M. Winslett Wilkins. On the use of an extended relational model to handle changing incomplete information. *Trans. on Software Engineering*, SE-11(7):620–633, July 1985.

22. R. Kowalski. Database updates in the event calculus. *Journal of Logic Programming*, 12:121–146, 1992.

23. R. Kowalski and F Sadri. The situation calculus and event calculus compared. Technical report, Department of Computing, Imperial College, London, England, 1994.

24. Yves Lespérance, Hector Levesque, Fangzhen Lin, Daniel Marcu, Raymond Reiter, and Richard Scherl. A logical approach to high-level robot programming – a progress report. In *Control of the Physical World by Intelligent Systems, Working Notes of the 1994 AAAI Fall Symposium*, November, 1994. New Orleans, LA.

25. H.L. Levesque, F. Lin, and R. Reiter. Defining complex actions in the situation calculus. Technical report, Dept. of Computer Science, Univ. of Toronto, 1995. In preparation.

26. 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.

27. F. Lin and R. Reiter. How to progress a database II: The STRIPS connection. Technical report, Department of Computer Science, University of Toronto, 1993.

28. F. Lin and R. Reiter. How to progress a database (and why) I. Logical foundations. In Jon Doyle, Erik Sandewall, and Pietro Torasso, editors, *Proc. KR'94, Fourth Int. Conf. on Principles of Knowledge Representation and Reasoning*, pages 425–436, 1994.
29. F. Lin and R. Reiter. State constraints revisited. *J. of Logic and Computation, special issue on actions and processes*, 4:655–678, 1994.
30. F. Lin and Y. Shoham. Provably correct theories of action. In *Proceedings of the National Conference on Artificial Intelligence*, 1991.
31. J.W. Lloyd. *Foundations of Logic Programming.* Springer Verlag, second edition, 1987.
32. 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.
33. J. McCarthy. Programs with common sense. In M. Minsky, editor, *Semantic Information Processing*, pages 403–418. The MIT Press, Cambridge, MA, 1968.
34. J. McCarthy. Epistemological problems of artificial intelligence. In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, pages 1038–1044, Cambridge, MA, 1977.
35. 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.
36. J. Minker, editor. *Foundations of Deductive Databases and Logic Programming.* Morgan Kaufmann Publishers, Inc., Los Altos, CA, 1988.
37. D. Poole. Explanation and prediction: an architecture for default and abductive reasoning. *Computational Intelligence*, 5:97–110, 1989.
38. 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.
39. R. Reiter. A sound and sometimes complete query evaluation algorithm for relational databases with null values. *Journal of the ACM*, 33(2):349–370, 1986.
40. 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.
41. R. Reiter. What should a database know? *Journal of Logic Programming*, 14(1-2):127–153, 1992.
42. R. Reiter. Proving properties of states in the situation calculus. *Artificial Intelligence*, 64:337–351, 1993.
43. R. Reiter. A simple solution to the frame problem (sometimes). Technical report, Department of Computer Science, University of Toronto, in preparation.
44. R. Reiter. Formalizing database evolution in the situation calculus. In *Proc. Fifth Generation Computer Systems*, pages 600–609, Tokyo, June 1 - 5, 1992.
45. R. Reiter. The projection problem in the situation calculus: A soundness and completeness result, with an application to database updates. In James Hendler, editor, *Proc. First Int. Conf. on Artificial Intelligence Planning Systems*, pages 198–203, College Park, Md., June 15 - 17, 1992. Morgan Kaufmann Publishers, Inc.
46. R. Reiter. On formalizing database updates: preliminary report. In *Proc. 3rd International Conference on Extending Database Technology*, pages 10–20, Vienna, March 23 - 27, 1992.
47. R. Waldinger. Achieving several goals simultaneously. In E. Elcock and D. Michie, editors, *Machine Intelligence 8*, pages 94–136. Ellis Horwood, Edinburgh, Scotland,

1977.

48. M. Winslett. Reasoning about action using a possible models approach. In *Proceedings of the National Conference on Artificial Intelligence*, pages 89–93, 1988.