Formalizing Database Evolution in the Situation Calculus

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

Abstract

We continue our exploration of a theory of database updates (Reiter [21, 23]) based upon the situation calculus. The basic idea is to take seriously the fact that databases evolve in time, so that updatable relations should be endowed with an explicit state argument representing the current database state. Database transactions are treated as functions whose effect is to map the current database state into a successor state. The formalism is identical to that arising in the artificial intelligence planning literature and indeed, borrows shamelessly from those ideas.

Within this setting, we consider several topics, specifically:

1. A logic programming implementation of query evaluation.
2. The treatment of database views.
3. State constraints and the ramification problem.
4. The evaluation of historical queries.
5. An approach to indeterminate transactions.

1 Introduction

Elsewhere (Reiter [21, 23]), we have described how one may represent databases and their update transactions within the situation calculus (McCarthy [13]). The basic idea is to take seriously the fact that databases evolve in time, so that updatable relations should be endowed with an explicit state argument representing the current database state. Database transactions are treated as functions, and the effect of a transaction is to map the current database state into a successor state. The resulting formalism becomes identical to theories of planning in the AI literature (See, for example, (Reiter [18])).

Following a review of some of the requisite basic concepts and results, we consider several topics in this paper:

1. We sketch a logic programming implementation of the axioms defining a database under updates. While we give no proof of its correctness, we observe that under suitable assumptions, Clark completion axioms (Clark [3]) should yield such a proof.

2. We show how our approach can accommodate database views.

3. The so-called *ramification problem*, as defined in the AI planning literature, arises in specifying database updates. Roughly speaking, this is the problem of incorporating, in the axiom defining an update transaction, the indirect effects of the update as given by arbitrary state constraints. We discuss this problem in the database setting, and characterize its solution in terms of inductive entailments of the database.

4. An historical query is one that references previous database states. We sketch an approach to such queries which reduces their evaluation to evaluation in the initial database state, together with conventional list processing techniques on the list of those update transactions leading to the current database state.

5. The database axiomatization of this paper addresses only *determinate* transactions; roughly speaking, in the presence of complete information about the current database state, such a transaction determines a unique successor state. By appealing to some ideas of Haas ([7]) and Schubert ([24]), we indicate how to axiomatize indeterminate database transactions.

2 Preliminaries

This section reviews some of the basic concepts and results of (Reiter [23, 21, 19]) which provide the necessary background for presenting the material of this paper. These include a motivating example, a precise specification of the axioms used to formalize update transactions and databases, an induction axiom suitable for proving properties of database states, and a discussion of query evaluation.

2.1 The Basic Approach: An Example

In (Reiter [23]), the idea of representing databases and their update transactions within the situation calculus was illustrated with an example education domain, which we repeat here.

Relations

The database involves the following three relations:

1. $enrolled(st, course, s)$: Student st is enrolled in course $course$ when the database is in state s.

2. $grade(st, course, grade, s)$: The grade of student st in course $course$ is $grade$ when the database is in state s.

3. $prerequ(pre, course)$: pre is a prerequisite course for course $course$. Notice that this relation is state independent, so is not expected to change during the evolution of the database.

Initial Database State

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

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

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

$$(\forall p).prerequ(p, P300) \equiv p = P100 \vee p = M100,$$
$$(\forall p)\neg prerequ(p, C100),$$
$$\begin{aligned}&(\forall c).enrolled(Bill, c, S_0) \equiv \\ &\quad c = M100 \vee c = C100 \vee c = P200,\end{aligned}$$
$$enrolled(Mary, C100, S_0),$$
$$\neg enrolled(John, M200, S_0), \ldots$$
$$grade(Sue, P300, 75, S_0), \quad grade(Bill, M200, 70, S_0), \ldots$$
$$prerequ(M200, M100), \quad \neg prerequ(M100, C100), \ldots$$

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:

$$\begin{aligned}&Poss(register(st, c), s) \equiv \\ &\{(\forall p).prerequ(p, c) \supset (\exists g).grade(st, p, g, s) \wedge g \geq 50\}.\end{aligned}$$[1]

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

$$\begin{aligned}&Poss(change(st, c, g), s) \equiv \\ &\qquad (\exists g').grade(st, c, g', s) \wedge g' \neq g.\end{aligned}$$

[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.

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. In what follows, $do(a,s)$ denotes that database state resulting from performing the update transaction a when the database is in state s.

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

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

2.2 An Axiomatization of Updates

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

Unique Names Axioms for Transactions

For distinct transaction names T and T',

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

Identical transactions have identical arguments:

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

for each function symbol T denoting a transaction.

Unique Names Axioms for States

$$(\forall a, s) S_0 \neq do(a, s),$$
$$(\forall a, s, a', s').do(a, s) = do(a', s') \supset a = a' \wedge s = s'.$$

Definition: The Simple Formulas

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

1. $F(\vec{t}, s)$ and $F(\vec{t}, S_0)$ are simple whenever F is an updatable database relation, the $\vec{t}$ are terms, and s is a variable of sort *state*. [2]
2. Any equality atom is simple.
3. Any other atom with predicate symbol other than *Poss* is simple.
4. If S_1 and S_2 are simple, so are $\neg S_1$, $S_1 \wedge S_2$, $S_1 \vee S_2$, $S_1 \supset S_2$, $S_1 \equiv S_2$.
5. If S is simple, so are $(\exists x)S$ and $(\forall x)S$ whenever x is an individual variable not of sort *state*.

In short, the simple formulas are those first order formulas whose updatable database relations do not mention the function symbol *do*, and which do not quantify over variables of sort *state*.

Definition: Transaction Precondition Axiom

A transaction precondition axiom is a formula of the form

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

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

Definition: Successor State Axiom

A successor state axiom for an $(n+1)$-ary updatable database relation F is a sentence of the form

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

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

[2]For notational convenience, we assume that the last argument of an updatable database relation is always the (only) argument of sort *state*.

2.3 An Induction Axiom

There is a close analogy between the situation calculus 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, an axiomatization in the situation calculus is a theory in which each "natural number" s has arbitrarily many successors.[3] Just as an induction axiom is necessary to prove anything interesting about the natural numbers, so also is induction required to prove general properties of states. This section is devoted to formulating an induction axiom suitable for this task.

We begin by defining 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 action of which is possible in that state resulting from executing the transactions preceeding it in the sequence. Hence, $<$ should be the smallest binary relation on states such that:

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

This can be achieved with a second order sentence, as follows:

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

$$\begin{array}{l}(\forall s, s').s < s' \equiv \\ \quad (\forall P).\{[(\forall a, s_1).Poss(a, s_1) \supset P(s_1, do(a, s_1))] \land \\ \quad\quad [(\forall a, s_1, s_2).Poss(a, s_2) \land P(s_1, s_2) \supset \\ \quad\quad\quad P(s_1, do(a, s_2))]\} \\ \quad\quad\quad\quad \supset P(s, s').\end{array} \tag{1}$$

$$(\forall s, s')s \leq s' \equiv s < s' \lor s = s'. \tag{2}$$

Reiter [20] shows how these axioms entail the following induction axiom suitable for proving properties of states s when $S_0 \leq s$:

$$\begin{array}{l}(\forall W).\{W(S_0) \land \\ \quad [(\forall a, s).Poss(a, s) \land S_0 \leq s \land W(s) \supset W(do(a, s))]\} \\ \quad\quad \supset (\forall s).S_0 \leq s \supset W(s).\end{array} \tag{3}$$

[3]There could even be infinitely many successors whenever an action is parameterized by a real number, as for example $move(block, location)$.

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

Reiter [23, 20] provides an approach to database integrity constraints in which the concept of a database satisfying its constraints is defined in terms of inductive entailment from the database, using this and other axioms of induction for the situation calculus. In this paper, we shall find other uses for induction in connection with database view definitions (Section 4), the so-called *ramification problem* (Section 5), and historical queries (Section 6).

2.4 Databases Defined

In the sequel, unless otherwise indicated, we shall only consider background database axiomatizations $\mathcal{D}$ of the form:

$$\mathcal{D} = \mathit{less\text{-}axioms} \cup \mathcal{D}_{ss} \cup \mathcal{D}_{tp} \cup \mathcal{D}_{uns} \cup \mathcal{D}_{unt} \cup \mathcal{D}_{S_0}$$

where

- *less-axioms* are the axioms (1), (2) for $<$ and $\leq$.
- $\mathcal{D}_{ss}$ is a set of successor state axioms, one for each updatable database relation.
- $\mathcal{D}_{tp}$ is a set of transaction precondition axioms, one for each database transaction.
- $\mathcal{D}_{uns}$ 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 database updatable relations of a sentence of $\mathcal{D}_{S_0}$. See Section 2.1 for an example $\mathcal{D}_{S_0}$. Thus, no updatable database relation 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").

2.5 Querying a Database

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

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

has been 'executed', we must determine whether

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

Querying an evolving database is precisely the *temporal projection problem* in AI planning [8].[4]

Definition: A Regression Operator $\mathcal{R}$

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

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

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

The use of the regression operator $\mathcal{R}$ is a classical plan synthesis technique (Waldinger [25]). See also (Pednault [16, 17]). Regression corresponds to the operation of *unfolding* in logic programming. For the class of databases of this paper, Reiter [23, 19] provides a sound and complete query evaluator based on regression. In this paper, we shall have a different use for regression, in connection with defining database views (Section 4).

[4]This property of our axiomatization makes the resulting approach quite different than Kowalski's situation calculus formalization of updates [9], in which each database update is accompanied by the addition of an atomic formula to the theory axiomatizing the database.

3 Updates in the Logic Programming Context

It seems that our approach to database updates can be implemented in a fairly straightforward way as a logic program, thereby directly complementing the logic programming perspective on databases (Minker [15]). For example, the axiomatization of the education example of Section 2.1 has the following representation as clauses:

Successor State Axiom Translation:

$$
\begin{aligned}
&enrolled(st, c, do(register(st, c), s)) \\
&\quad \leftarrow Poss(register(st, c), s). \\
&enrolled(st, c, do(a, s)) \\
&\quad \leftarrow a \neq drop(st, c), enrolled(st, c, s), Poss(a, s). \\
&grade(st, c, g, do(change(st, c, g), s)) \\
&\quad \leftarrow Poss(change(st, c, g), s). \\
&grade(st, c, g, do(a, s)) \\
&\quad \leftarrow a \neq change(st, c, g'), grade(st, c, g, s), Poss(a, s).^{5}
\end{aligned}
$$

Transaction Precondition Axiom Translation:

$$
\begin{aligned}
&Poss(register(st, c), s) \leftarrow not\ P(st, c, s). \\
&Q(st, p, s) \leftarrow grade(st, p, g, s), g \geq 50.^{6} \\
&Poss(change(st, c, g), s) \leftarrow grade(st, c, g', s), g \neq g'. \\
&Poss(drop(st, c), s) \leftarrow enrolled(st, c, s).
\end{aligned}
$$

5 This translation is problematic because it invokes negation-as-failure on a non-ground atom. The intention is that whenever a is bound to a term whose function symbol is $change$, the call should fail. This can be realized procedurally by retaining the clause sequence as shown, and simply deleting the inequality $a \neq change(st, c, g')$.

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

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

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

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 [3] 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 [12], pp.79, 109).

4 Views

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

$$(\forall \vec{x}, s).V(\vec{x}, s) \equiv \mathcal{B}(\vec{x}, s), \tag{4}$$

where $\mathcal{B}$ is a simple formula with free variables among $\vec{x}$ and s, and which mentions only base predicates.[7] Unfortunately, sentences like (4) 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:

$$(\forall \vec{x}).V(\vec{x}, S_0) \equiv \mathcal{B}(\vec{x}, S_0), \tag{5}$$

$$\begin{aligned}&(\forall a, s).Poss(a, s) \supset \\ &\qquad (\forall \vec{x}).V(\vec{x}, do(a, s)) \equiv \mathcal{R}[\mathcal{B}(\vec{x}, do(a, s))].^{8}\end{aligned} \tag{6}$$

Sentence (5) is a perfectly good candidate for inclusion in $\mathcal{D}_{S_0}$, while (6) has the syntactic form of a successor state axiom and hence may be included in $\mathcal{D}_{ss}$.

[7]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.

[8]Notice that since we are not considering recursive views (i.e., $\mathcal{B}$ does not mention V), the formula $\mathcal{R}[\mathcal{B}(\vec{x}, do(a, s))]$ is well defined.

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

Theorem 1 *Suppose $V(\vec{x}, s)$ is an updatable database relation, and that $\mathcal{B}(\vec{x}, s)$ 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 (6) for V, and that $\mathcal{D}_{S_0}$ contains the initial state axiom (5). Then,*

$$\mathcal{D} \cup \{3\} \models (\forall s).S_0 \leq s \supset (\forall \vec{x}).V(\vec{x}, s) \equiv \mathcal{B}(\vec{x}, s).$$

Theorem 1 informs us that from the initial state and successor state axioms (5) and (6) 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 (4) 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 (4) will be true. We take this as sufficient justification for representing views within our framework by the axioms (5) and (6).

5 State Constraints and the Ramification Problem

Recall that our definition of a database (Section 2.4) 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 $C100$.

$$\begin{gathered}(\forall s, st).S_0 \leq s \wedge enrolled(st, C200, s) \supset \\ enrolled(st, C100, s).\end{gathered} \tag{7}$$

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) 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. Reiter [20] proposes that dynamic integrity constraints should be viewed as inductive entailments of the database, and gives several examples of such derivations. Moreover, Theorem 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 (5) and the successor state axiom (6).

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

$$(\forall s_1, \ldots, s_n).S_0 \leq s_i \wedge s_i \leq s_j \wedge \cdots \supset W(s_1, \ldots, s_n),$$

and that a database is said to *satisfy* this constraint iff the database inductively entails it.[9]

The fact that state constraints like (7) 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 2.1) was:

$$\begin{aligned} &Poss(a, s) \supset \{enrolled(st, c, do(a, s)) \equiv \\ &\quad a = register(st, c) \vee \\ &\qquad enrolled(st, c, s) \wedge a \neq drop(st, c)\}. \end{aligned} \tag{8}$$

As one would expect, this does not inductively entail (7). To accommodate the state constraint (7), this successor state axiom must be changed to:

$$\begin{aligned} &Poss(a, s) \supset \{enrolled(st, c, do(a, s)) \equiv \\ &a = register(st, c) \wedge [c = C200 \supset enrolled(st, C100, s)] \\ &\vee \\ &enrolled(st, c, s) \wedge a \neq drop(st, c) \wedge \\ &\qquad [c = C200 \supset a \neq drop(st, C100)]\}. \end{aligned} \tag{9}$$

[9]See Section 2.3 for a brief discussion of inductively proving properties of states in the situation calculus.

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),

$$enrolled(st, C200, S_0) \supset enrolled(st, C100, S_0),$$

then (7) is an inductive entailment of the database.

The example illustrates the subtleties involved in getting the successor state axioms to reflect the intent of a state constraint. These difficulties are a manifestation of the so-called *ramification problem* in artificial intelligence planning domains (Finger [4]). Transactions might have ramifications, or *indirect effects*. For the example at hand, the transaction of registering a student in $C200$ has 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 (9) of (8) was designed to capture this indirect effect. In our setting, the ramification problem is this: Given a static state constraint like (7), 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? A variety of circumscriptive proposals for addressing the ramification problem have been proposed in the artificial intelligence literature, notably by Baker [1], Baker and Ginsberg [2], Ginsberg and Smith [5], Lifschitz [10] and Lin and Shoham [11]. Our formulation of the problem in terms of inductive entailments of the database seems to be new. For the databases of this paper, Fanghzen Lin[10] appears to have a solution to this problem.

6 Historical Queries

Using the relations $<$ and $\leq$ on states, as defined in Section 2.3, it is possible to pose *historical* queries to a database. First, some notation.

Notation: $do([a_1, \ldots, n], s)$

Let $a_1, \ldots, a_n$ be transactions. Define

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

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

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

[10] Personal communication.

$do([a_1, \ldots, a_n], s)$ 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.

Now, suppose $\mathbf{T}$ 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 the database was ever in a state in which John was simultaneously enrolled in both $C100$ and $M100$?

$$(\exists s).S_0 \leq s \land s \leq do(\mathbf{T}, S_0) \land \\ enrolled(John, C100, s) \land enrolled(John, M100, s). \tag{10}$$

Has Sue always worked in department 13?

$$(\forall s).S_0 \leq s \land s \leq do(\mathbf{T}, S_0) \supset emp(Sue, 13, s). \tag{11}$$

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 list of those transactions leading to the current database state.

Begin by considering two new predicates, *last* and *mem-diff*. The intended interpretation of $last(s, a)$ is that the transaction a is the last transaction of the sequence s. For example,

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

is true, while

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

is false, assuming unique names axioms for transactions. The following two axioms are sufficient for our purposes:

$$\neg(last(S_0, a).$$

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

The intended interpretation of $mem\text{-}diff(a,s,s')$ is that transaction a is a member of the "list difference" of s and s', where state s' is a "sublist" of s. For example,

$$\begin{array}{l} mem\text{-}diff(drop(Mary, C100), \\ \quad do([register(John, C100), drop(Bill, C100), \\ \quad\quad drop(Mary, C100), drop(John, M100)], S_0), \\ \quad do([register(John, C100)], S_0)) \end{array}$$

is true, whereas

$$\begin{array}{l} mem\text{-}diff(register(Mary, C100), \\ \quad do([register(John, C100), drop(Bill, C100), \\ \quad\quad drop(Mary, C100), drop(John, M100)], S_0), \\ \quad do([register(John, C100)], S_0)) \end{array}$$

is false (assuming unique names axioms for transactions). The following axioms will be sufficient for our needs:

$$\neg mem\text{-}diff(a,s,s).$$

$$s \leq s' \supset mem\text{-}diff(a, do(a,s'), s).$$

$$mem\text{-}diff(a,s,s') \supset mem\text{-}diff(a, do(a',s), s').$$

$$mem\text{-}diff(a, do(a',s), s') \wedge a \neq a' \supset mem\text{-}diff(a,s,s').$$

We begin by showing how to answer query (11). Suppose, for the sake of the example, that the successor state axiom for emp is:

$$\begin{array}{l} Poss(a,s) \supset emp(p,d,do(a,s)) \equiv a = hire(p,d) \vee \\ \quad emp(p,d,s) \wedge a \neq fire(p) \wedge a \neq quit(p). \end{array}$$

Using this, and the sentences for $last$ and $mem\text{-}diff$ together with the induction axiom (3), it is possible to prove:

$$\begin{array}{l} S_0 \leq s \supset emp(p,d,s) \equiv emp(p,d,S_0) \wedge \\ \neg mem\text{-}diff(fire(p), s, S_0) \wedge \neg mem\text{-}diff(quit(p), s, S_0) \vee \\ (\exists s').S_0 \leq s' \leq s \wedge last(s', hire(p,d)) \wedge \\ \neg mem\text{-}diff(fire(p), s, s') \wedge \neg mem\text{-}diff(quit(p), s, s'). \end{array}$$

Using this and the (reasonable) assumption that the transaction sequence **T** is legal,[11] it is simple to prove that the query (11) is equivalent to:

$$\left\{\begin{array}{l} emp(Sue, 13, S_0) \wedge \\ \neg \mathit{mem\text{-}diff}(fire(Sue), do(\mathbf{T}, S_0), S_0) \wedge \\ \neg \mathit{mem\text{-}diff}(quit(Sue), do(\mathbf{T}, S_0), S_0) \end{array}\right\}$$

$$\vee$$

$$\left\{\begin{array}{l} (\exists s').S_0 \leq s' \leq do(\mathbf{T}, S_0) \wedge \\ last(s', hire(Sue, 13)) \wedge \\ \neg \mathit{mem\text{-}diff}(fire(Sue), do(\mathbf{T}, S_0), s') \wedge \\ \neg \mathit{mem\text{-}diff}(quit(Sue), do(\mathbf{T}, S_0), s'). \end{array}\right\}$$

This form of the original query is of interest because it reduces query evaluation to evaluation in the initial database state, together with simple *list processing* on the list **T** of those transactions leading to the current database state. We can verify that *Sue* has always been employed in department 13 in one of two ways:

1. Verify that she was initially employed in department 13, and that neither $fire(Sue)$ nor $quit(Sue)$ are members of list **T**.

2. Verify that **T** has a sublist ending with $hire(Sue, 13)$, and that neither $fire(Sue)$ nor $quit(Sue)$ are members of the list difference of **T** and this sublist.[12]

We now consider evaluating the first query (10) in the same list processing spirit. We shall assume that (8) is the successor state axiom for *enrolled*. Using the above sentences for *last* and *mem-diff*, together with (8) and the induction axiom (3), it is possible to prove:

$$\begin{array}{l} S_0 \leq s \supset enrolled(st, c, s)) \equiv \\ \quad enrolled(st, c, S_0) \wedge \neg \mathit{mem\text{-}diff}(drop(st, c), s, S_0) \vee \\ \quad (\exists s').S_0 \leq s' \leq s \wedge last(s', register(st, c)) \wedge \\ \qquad\qquad \neg \mathit{mem\text{-}diff}(drop(st, c), s, s'). \end{array}$$

[11]Intuitively, **T** is legal iff each transaction of **T** satisfies its preconditions (see Section 2.1) in that state resulting from performing all the transactions preceeding it in the sequence, beginning with state S_0. See (Reiter [19]) for details, and a procedure for verifying the legality of a transaction sequence.

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

Then, on the assumption that the transaction sequence **T** is legal, it is simple to prove that the query (10) is equivalent to:

$$
\begin{array}{l}
(\exists s).S_0 \leq s \leq do(\mathbf{T}, S_0) \wedge \\
\Big[\\
\quad \left\{ \begin{array}{l} enrolled(John, C100, S_0) \wedge \\ enrolled(John, M100, S_0) \wedge \\ \neg mem\text{-}diff(drop(John, C100), s, S_0) \wedge \\ \neg mem\text{-}diff(drop(John, M100), s, S_0) \end{array} \right\} \\
\vee \\
\quad \left\{ \begin{array}{l} enrolled(John, C100, S_0) \wedge \\ \neg mem\text{-}diff(drop(John, C100), s, S_0) \wedge \\ (\exists s').S_0 \leq s' \leq s \wedge \\ last(s', register(John, M100)) \wedge \\ \neg mem\text{-}diff(drop(John, M100), s, s') \end{array} \right\} \\
\vee \\
\quad \left\{ \begin{array}{l} enrolled(John, M100, S_0) \wedge \\ (\exists s'').S_0 \leq s'' \leq s \wedge \\ last(s'', register(John, C100)) \wedge \\ \neg mem\text{-}diff(drop(John, C100), s, s'') \end{array} \right\} \\
\vee \\
\quad \left\{ \begin{array}{l} (\exists s', s'').S_0 \leq s' \leq s \wedge S_0 \leq s'' \leq s \wedge \\ last(s', register(John, M100)) \wedge \\ last(s'', register(John, C100)) \wedge \\ \neg mem\text{-}diff(drop(John, M100), s, s') \wedge \\ \neg mem\text{-}diff(drop(John, C100), s, s'') \end{array} \right\} \\
\Big]
\end{array}
$$

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 as follows. Find a sublist (loosely denoted by s) of **T** such that one of the following four conditions holds:

1. *John* was initially enrolled in both $C100$ and $M100$ and neither $drop(John, C100)$ nor $drop(John, M100)$ are members of list s.

2. *John* was initially enrolled in $C100$, $drop(John, C100)$ is not a member of list s, s has a sublist s' ending with $register(John, M100)$ and

$drop(John, M100)$ is not a member of the list difference of s and s'.

3. $John$ was initially enrolled in $M100$, $drop(John, M100)$ is not a member of list s, s has a sublist s' ending with $register(John, C100)$ and $drop(John, C100)$ is not a member of the list difference of s and s'.

4. There are two sublists s' and s'' of s, s' ends with $register(John, M100)$, s'' ends with $register(John, C100)$, $drop(John, M100)$ is not a member of the list difference of s and s', and $drop(John, C100)$ is not a member of the list difference of s and s''.

We can even pose queries about the future, for example, is it possible for the database ever to be in a state in which $John$ is enrolled in both $C100$ and $C200$?

$$(\exists s).S_0 \leq s \wedge enrolled(John, C100, s) \wedge \\ enrolled(John, C200, s).$$

Answering queries of this form is precisely the problem of plan synthesis in AI (Green [6]). For the class of databases of this paper, Reiter [22, 18] shows how regression provides a sound and complete evaluator for such queries.

7 Indeterminate Transactions

A limitation of our formalism is that it requires all transactions to be *determinate*, by which we mean that in the presence of complete information about the initial database state a transaction completely determines the resulting state.

One way to extend the theory to include indeterminate transactions is by appealing to a simple idea due to Haas [7], as elaborated by Schubert [24]. As an example, consider the indeterminate transaction $drop\text{-}a\text{-}student(c)$, meaning that some student – we don't know whom – is to be dropped from course c. Notice that we cannot now have a successor state axiom of the form

$$Poss(a, s) \supset \{enrolled(st, c, do(a, s)) \equiv \Phi(st, c, a, s)\}.$$

To see why, consider the following instance of this axiom:

$$\begin{aligned}&Poss(\textit{drop-a-student(C100)}, S_0) \supset \\ &\quad \{enrolled(John, C100, do(\textit{drop-a-student(C100)}, S_0)) \\ &\qquad \equiv \Phi(John, C100, \textit{drop-a-student(C100)}, S_0)\}.\end{aligned}$$

Suppose Σ_0 is a complete description of the initial database state, and suppose moreover, that

$$\begin{aligned}\Sigma_0 \models\ & Poss(\textit{drop-a-student(C100)}, S_0) \wedge \\ & enrolled(John, C100, S_0).\end{aligned}$$

By the completeness assumption,

$$\Sigma_0 \models \pm\Phi(John, C100, \textit{drop-a-student(C100)}, S_0),$$

in which case

$$\begin{aligned}\Sigma_0 \models \pm & enrolled(John, C100, \\ & do(\textit{drop-a-student(C100)}, S_0)).\end{aligned}$$

In other words, we would know whether $John$ was the student dropped from $C100$, violating the intention of the *drop-a-student* transaction.

Despite the inadequacies of the axiomatization of Section 2.2 (specifically the failure of successor state axioms for specifying indeterminate transactions), we can represent this setting with something like the following axioms:

$$(\exists st) enrolled(st, c, s) \supset Poss(\textit{drop-a-student}(c), s).$$

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

$$\begin{aligned}&Poss(a, s) \supset \\ &\qquad \{a = drop(st, c) \supset \neg enrolled(st, c, do(a, s))\}.\end{aligned}$$

$$\begin{aligned}&Poss(a, s) \supset \{a = \textit{drop-a-student}(c) \supset \\ &\quad (\exists! st) enrolled(st, c, s) \wedge \neg enrolled(st, c, do(a, s)\}.^{13}\end{aligned}$$

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

[13] $(\exists! st)$ denotes the existence of a *unique* st.

$$
\begin{array}{l}
Poss(a, s) \supset \\
\quad \{enrolled(st, c, s) \wedge \neg enrolled(st, c, do(a, s)) \supset \\
\quad\quad a = drop(st, c) \vee a = drop\text{-}a\text{-}student(c)\}.
\end{array}
$$

The last two formulas are examples of what Schubert [24] calls *explanation closure axioms.* For the example at hand, the last axiom provides an exhaustive enumeration of those transactions (namely $drop(st, c)$ and $drop\text{-}a\text{-}student(c)$) which could possibly explain how it came to be that st is enrolled in c in the current state s and is not enrolled in c in the successor state. Similarly, the second last axiom explains how a student could come to be enrolled in a course in which she was not enrolled previous to the transaction.[14] The feasibility of such an approach relies on a closure assumption, namely that we, as database designers, can provide a finite exhaustive enumeration of such explaining transactions.[15] In the "real" world, such a closure assumption is problematic. The state of the world has changed so that a student is no longer enrolled in a course. What can explain this? The school burned down? The student was kidnapped? The teacher was beamed to Andromeda by extraterrestrials? Fortunately, in the database setting, such open-ended possible explaining events are precluded by the database designer, by virtue of her initial choice of some closed set of transactions with which to model the application at hand; no events outside this closed set (school burned down, student kidnapped, etc.) can be considered in defining the evolution of the database. This initial choice of a closed set of transactions having been made, explanation closure axioms provide a natural representation of this closure assumption.

By appealing to explanation closure axioms, we can now specify indeterminate transactions. The price we pay is the loss of the simple regression-based query evaluator of (Reiter [23, 21]); we no longer have a simple sound and complete query evaluator. Of course, conventional first order theorem-proving does provide a query evaluator for such an axiomatization. For example, the following are entailments of the above axioms, together with

[14]It is these explanation closure axioms which provide a succinct alternative to the frame axioms (McCarthy and Hayes [14]) which would normally be required to represent dynamically changing worlds like databases (Reiter [23]).

[15]This assumption is already implicit in our successor state axioms of Section 2.2

unique names axioms for transactions and for $John$ and $Mary$:

$$\begin{array}{l} enrolled(John, C100, S_0) \wedge enrolled(Mary, C100, S_0) \\ \quad \supset \\ \quad enrolled(John, C100, do(drop(Mary, C100), S_0)) \wedge \\ \quad \neg enrolled(Mary, C100, do(drop(Mary, C100), S_0)). \end{array}$$

$$\begin{array}{l} \{(\forall st).enrolled(st, C100, S_0) \equiv st = John\} \supset \\ \qquad (\forall st)\neg enrolled(st, C100, \\ \qquad\qquad do(drop\text{-}a\text{-}student\ (C100), S_0)). \end{array}$$

$$\begin{array}{l} \{(\forall st).enrolled(st, C100, S_0) \equiv \\ \qquad\qquad st = John \vee st = Mary\} \\ \quad \supset \\ enrolled(John, C100, do(drop\text{-}a\text{-}student(C100), S_0)) \oplus \\ enrolled(Mary, C100, do(drop\text{-}a\text{-}student(C100), S_0)). \end{array}$$

Notice that the induction axiom (3) of Section 2.3 does not depend on any assumptions about the underlying database. In particular, it does not depend on successor state axioms. It follows that we can continue to use induction to prove properties of database states and integrity constraints in the more generalized setting of indeterminate transactions. The fundamental perspective on integrity constraints of (Reiter [20]) – namely that they are inductive entailments of the database – remains the same.

Acknowledgements

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

References

[1] 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.

[2] 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.

[3] 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.

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

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

[6] 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.

[7] A. R. Haas. The case for domain-specific frame axioms. In F. M. Brown, editor, *The frame problem in artificial intelligence. Proceedings of the 1987 workshop*, pages 343–348, Los Altos, California, 1987. Morgan Kaufmann Publishers, Inc.

[8] 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.

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

[10] 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.

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

[12] J.W. Lloyd. *Foundations of Logic Programming.* Springer Verlag, second edition, 1987.

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

[14] 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.

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

[16] E.P.D. Pednault. Synthesizing plans that contain actions with context-dependent effects. *Computational Intelligence*, 4:356–372, 1988.

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

[18] 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.

[19] R. Reiter. The projection problem in the situation calculus: A soundness and completeness result, with an application to database updates. 1992. submitted for publication.

[20] R. Reiter. Proving properties of states in the situation calculus. 1992. submitted for publication.

[21] R. Reiter. On specifying database updates. Technical report, Department of Computer Science, University of Toronto, in preparation.

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

[23] R. Reiter. On formalizing database updates: preliminary report. In *Proc. 3rd International Conference on Extending Database Technology*, Vienna, March 23 - 27, 1992. to appear.

[24] L.K. Schubert. Monotonic solution of the frame problem in the situation calculus: an efficient method for worlds with fully specified actions. In H.E. Kyberg, R.P. Loui, and G.N. Carlson, editors, *Knowledge Representation and Defeasible Reasoning*, pages 23–67. Kluwer Academic Press, 1990.

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