How to Progress a Database (and Why)
I. Logical Foundations

Fangzhen Lin and Ray Reiter*
Department of Computer Science
University of Toronto
Toronto, Canada M5S 1A4
email: fl@ai.toronto.edu reiter@ai.toronto.edu

Abstract

One way to think about STRIPS is as a mapping from databases to databases, in the following sense: Suppose we want to know what the world would be like if an action, represented by the STRIPS operator α, were done in some world, represented by the STRIPS database $\mathcal{D}_0$. To find out, simply perform the operator α on $\mathcal{D}_0$ (by applying α's elementary *add* and *delete* revision operators to $\mathcal{D}_0$). We describe this process as *progressing the database* $\mathcal{D}_0$ in response to the action α. In this paper, we consider the general problem of progressing an initial database in response to a given sequence of actions. We appeal to the situation calculus and an axiomatization of actions which addresses the frame problem (Reiter [13], Lin and Reiter [8]). This setting is considerably more general than STRIPS. Our results concerning progression are mixed. The (surprising) bad news is that, in general, to characterize a progressed database we must appeal to second order logic. The good news is that there are many useful special cases for which we can compute the progressed database in first order logic; not only that, we can do so efficiently.

1 INTRODUCTION

One way to think about STRIPS is as a mapping from databases to databases, in the following sense: Suppose we want to know what the world would be like if an action, represented by the STRIPS operator α, were done in some world, represented by the STRIPS database $\mathcal{D}_0$. To find out, simply perform the operator α on $\mathcal{D}_0$ (by applying α's elementary *add* and *delete* revision operators to $\mathcal{D}_0$). We describe this process as *progressing the database* $\mathcal{D}_0$ in response to the action α (cf. Rosenschein [15] and Pednault [9]). The resulting database describes the effects of the action on the world represented by the initial database.[1] However, it may not always be convenient or even possible to describe the effects of actions as a simple process of progressing an initial world description. As we shall see in this paper, once we go beyond STRIPS-like systems, progression becomes surprisingly complicated.

In this paper, we consider the general problem of progressing an initial database in response to a given sequence of actions. We appeal to the situation calculus and an axiomatization of actions which addresses the frame problem (Reiter [13], Lin and Reiter [8]). This setting is considerably more general than STRIPS. Our results concerning progression are mixed. The (surprising) bad news is that, in general, to characterize a progressed database we must appeal to second order logic. The good news is that there are many useful special cases for which we can compute the progressed database in first order logic; not only that, we can do so efficiently.

The need to progress a database arises for us in a robotics setting. In our approach to controlling a robot,[2] we must address the so-called *projection problem*: Answer the query $Q(do(\mathbf{A}, S_0))$, where $do(\mathbf{A}, S_0)$ denotes the situation resulting from performing the sequence of actions $\mathbf{A}$ beginning with the initial situation S_0. This can be done using regression (cf. Waldinger [17], Pednault [10], and Reiter [12]) to reduce the projection problem to one of entailment from the *initial* database, consisting of sentences about the initial situation S_0. Unfortunately, regression suffers from a number of drawbacks in this application:

*Fellow of the Canadian Institute for Advanced Research

[1] This is also the way that database practitioners think about database updates (Abiteboul [1]). In fact, the STRIPS action and database update paradigms are essentially the same. Accordingly, this paper is as much about database updates as it is about STRIPS actions and their generalizations.

[2] Joint work with Yves Lespérance, Hector Levesque, Bill Millar and Richard Scherl.

1. After the robot has been functioning for a long time, the sequence $\mathbf{A}$, consisting of all the actions it has performed since the initial situation, is extremely long, and regressing over such a long sequence becomes a computational burden.
2. Similarly, after a long while, the world state often becomes so rearranged that significantly many final steps of the regression become entirely unnecessary.
3. Most significantly, for robotics, *perceptual actions* (Scherl and Levesque [16]) lead to new facts being added to the database. But such facts are true in the current situation – the one immediately following the perceptual action – whereas the other (old) database facts are true in S_0. Reasoning about databases containing mixed facts – facts about the current and initial situations – is very complicated, and we know of no satisfactory way to do this.

Our way of addressing these problems with regression is to to periodically progress the robot's database. In particular, every perceptual action is accompanied by a progression of the database, coupled with the addition of the perceived fact to the resulting database. We envisage that these database progression computations can be done off-line, during the time when the robot is busy performing physical actions, like moving about.

2 LOGICAL PRELIMINARIES

The language $\mathcal{L}$ of the situation calculus is a many-sorted first-order one with the sorts *state* for situations, *action* for actions, and *object* for anything else. We have the following domain independent predicates and functions: a unique constant S_0 of sort *state*; a binary function $do(a,s)$ that denotes the state resulting from performing the action a in the state s; a binary predicate $Poss(a,s)$ that expresses the conditions for the action a to be executable in the state s; and a binary predicate $<$: $state \times state$. We shall follow convention, and write $<$ in infix form. By $s < s'$ we mean that s' can be obtained from s by a sequence of executable actions. As usual, $s \leq s'$ will be a shorthand for $s < s' \vee s = s'$. We assume a finite number of *state independent* predicates which are the ones with arity $object^n$, $n \geq 0$, a finite number of *state independent* functions which are the ones with arity $object^n \rightarrow object$, $n \geq 0$, and a finite number of *fluents* which are predicate symbols of arity $object^n \times state$, $n \geq 0$.

We shall denote by $\mathcal{L}^2$ the second-order extension of $\mathcal{L}$. Our foundational axioms for the situation calculus will be in $\mathcal{L}^2$ (Lin and Reiter [8]), because we need induction on situations (Reiter [14]).

We shall frequently need to restrict the situation calculus to a particular situation. For instance, the initial database is defined to be a finite set of sentences in $\mathcal{L}$ that do not mention any state terms except S_0, and do not mention $Poss$ and $<$. For this purpose, for any state term st, we shall define $\mathcal{L}_{st}$ to be the subset of $\mathcal{L}$ that does not mention any other state terms except st, does not quantify over state variables, and does not mention $Poss$ and $<$. Formally, it is the smallest set satisfying

1. $\varphi \in \mathcal{L}_{st}$ provided $\varphi \in \mathcal{L}$ does not mention any state term.
2. $F(t_1,...,t_n,st) \in \mathcal{L}_{st}$ provided F is a fluent of the right arity, and $t_1,...,t_n$ are terms of the right sort.
3. If φ and φ' are in $\mathcal{L}_{st}$, so are $\neg\varphi$, $\varphi \vee \varphi'$, $\varphi \wedge \varphi'$, $\varphi \supset \varphi'$, $\varphi \equiv \varphi'$, $(\forall x)\varphi$, $(\exists x)\varphi$, $(\forall a)\varphi$, and $(\exists a)\varphi$, where x and a are variables of sort *object* and *action*, respectively.

We remark here that according to this definition, $(\forall a)F(do(a,S_0))$ will be in $\mathcal{L}_{do(a,S_0)}$. This may seem odd when we want sentences in $\mathcal{L}_{st}$ to be propositions about situation st. Fortunately, we shall use $\mathcal{L}_{st}$ only when st is either a ground term or a simple variable of sort *state*.

We shall use $\mathcal{L}^2_{st}$ to denote the second-order extension of $\mathcal{L}_{st}$ by predicate variables of arity $object^n$, $n \geq 0$. So the second-order sentence $(\exists p)(\forall x).p(x) \equiv F(x,S_0)$ is in $\mathcal{L}^2_{S_0}$, but $(\exists p)(\forall x)(\exists s).p(x,s) \equiv F(x,S_0)$ is not, since the latter quantifies over a predicate variable of arity $object \times state$. Formally, $\mathcal{L}^2_{st}$ is the smallest set satisfying

1. Every formula in $\mathcal{L}_{st}$ is also in $\mathcal{L}^2_{st}$.
2. $p(t_1,...,t_n) \in \mathcal{L}^2_{st}$ provided p is a predicate variable of arity $object^n$, $n \geq 0$, and $t_1,...,t_n$ are terms of sort *object*.
3. If φ and φ' are in $\mathcal{L}^2_{st}$, so are $\neg\varphi$, $\varphi \vee \varphi'$, $\varphi \wedge \varphi'$, $\varphi \supset \varphi'$, $\varphi \equiv \varphi'$, $(\forall x)\varphi$, $(\exists x)\varphi$, $(\forall a)\varphi$, $(\exists a)\varphi$, $(\forall p)\varphi$, and $(\exists p)\varphi$, where x and a are variables of sort *object* and *action*, respectively, and p is a predicate variable of arity $object^n$, $n \geq 0$.

3 BASIC ACTION THEORIES

We assume that our action theory $\mathcal{D}$ has the following form (cf. Reiter [13] and Lin and Reiter [8]):

$$\mathcal{D} = \Sigma \cup \mathcal{D}_{ss} \cup \mathcal{D}_{ap} \cup \mathcal{D}_{una} \cup \mathcal{D}_{S_0},$$

where

- Σ, given below, is the set of the foundational axioms for situations.

- $\mathcal{D}_{ss}$ is a set of successor state axioms of the form:[3]

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

where F is a fluent, and $\Phi_F(\vec{x}, a, s)$ is in $\mathcal{L}_s$.

- $\mathcal{D}_{ap}$ is a set of action precondition axioms of the form:

$$Poss(A(\vec{x}), s) \equiv \Psi_A(\vec{x}, s),$$

where A is an action, and $\Psi_A(\vec{x}, s)$ is in $\mathcal{L}_s$.

- $\mathcal{D}_{una}$ is the set of unique names axioms for actions: For any two different actions $A(\vec{x})$ and $A'(\vec{y})$, we have

$$A(\vec{x}) \neq A'(\vec{y}),$$

and for any action $A(x_1, ..., x_n)$, we have

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

- $\mathcal{D}_{S_0}$, the initial database, is a finite set of first-order sentences in $\mathcal{L}_{S_0}$.

We shall give an example of our action theory in a moment. First, we briefly explain our foundational axioms Σ since they are independent of particular applications. Σ contains axioms about the structure of situations. Formally, Σ is the following set of axioms:

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

$$do(a_1, s_1) = do(a_2, s_2) \supset (a_1 = a_2 \wedge s_1 = s_2),$$

$$(\forall P)[P(S_0) \wedge (\forall a, s)(P(s) \supset P(do(a, s))) \supset (\forall s)P(s)],$$

$$\neg s < S_0,$$

$$s < do(a, s') \equiv (Poss(a, s') \wedge s \leq s').$$

Notice the similarity between Σ and Peano Arithmetic. The first two axioms are unique names assumptions. They eliminate finite cycles, and merging. The third axiom is second-order induction. It amounts to the domain closure axiom which says that every situation has to be obtained from repeatedly applying do to S_0.[4] The last two axioms define $<$ inductively.

Σ is the only place where axioms about the structure of situations can appear. It is needed only if we want to show, usually by induction, that a state constraint of the form $(\forall s).C(s)$ is entailed by an action theory. For the purpose of temporal projection, in particular progression as we shall see, $\mathcal{D}$ has exactly the same effect as $\mathcal{D} - \Sigma$: For any formula $\varphi(s)$ in $\mathcal{L}_s$, and any sequence $\mathbf{A}$ of actions,

$$\mathcal{D} \models \varphi(do(\mathbf{A}, S_0))$$

iff

$$\mathcal{D}_{ss} \cup \mathcal{D}_{ap} \cup \mathcal{D}_{una} \cup \mathcal{D}_{S_0} \models \varphi(do(\mathbf{A}, S_0)).$$

This follows directly from the following proposition which will be used throughout this paper.

Proposition 3.1 *Given any model M of $\mathcal{D}_{ss} \cup \mathcal{D}_{ap} \cup \mathcal{D}_{una} \cup \mathcal{D}_{S_0}$, there is a model M' of $\mathcal{D}$ such that:*

1. *M' and M have the same domains for sorts* action *and* object*, and interpret all state independent predicates and functions the same;*
2. *For any sequence $\mathbf{A}$ of actions, any fluent F, and any variable assignment σ:*

$$M', \sigma \models F(\vec{x}, do(\mathbf{A}, S_0))$$

iff

$$M, \sigma \models F(\vec{x}, do(\mathbf{A}, S_0)).$$

Example 3.1 An educational database (Reiter [13]). There are two fluents:

- $enrolled(st, course, s)$: student st is enrolled in course $course$ in state s.
- $grade(st, course, grade, s)$: the grade of st in $course$ is $grade$ in state s.

There are two state independent predicates:

- $prerequ(pre, course)$: pre is a prerequisite course for course $course$.
- $better(grade1, grade2)$: grade $grade1$ is better than grade $grade2$.

There are three database transactions:

- $register(st, course)$: register the student st into course $course$.
- $change(st, course, grade)$: change the grade of the student st in course $course$ to $grade$.
- $drop(st, course)$: drop the student st from course $course$.

This setting can be axiomatized as follows.

$\mathcal{D}_{ss}$ is the set of following successor state axioms:

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

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

[3] In the following, unless otherwise stated, all free variables in a formula are assumed to be universally quantified from the outside.

[4] For a discussion of the use of induction in the situation calculus, see (Reiter [14]).

$\mathcal{D}_{ap}$ is the set of following action precondition axioms:

$$Poss(register(st,c),s) \equiv (\forall pr).prerequ(pr,c) \supset (\exists g).grade(st,pr,g,s) \wedge better(g,50),$$

$$Poss(change(st,c,g),s) \equiv True,$$

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

$\mathcal{D}_{S_0}$, the initial database, can be any finite set of axioms about the initial state, for example, the following ones:

$$John \neq Sue \neq C100 \neq C200 \wedge prerequ(C100, C200),$$

$$enrolled(Sue, C100, S_0),$$

$$enrolled(John, C100, S_0) \vee enrolled(John, C200, S_0).$$

■

4 FORMAL FOUNDATIONS

Let α be a ground simple action, e.g. $enroll(Sue, C100)$, and let S_α denote the state term $do(\alpha, S_0)$. A progression $\mathcal{D}_{S_\alpha}$ of $\mathcal{D}_{S_0}$ in response to α should have the following properties:

1. $\mathcal{D}_{S_\alpha}$ is a set of sentences about state S_α only, i.e., in $\mathcal{L}_{S_\alpha}$ or in $\mathcal{L}^2_{S_\alpha}$.
2. For all queries about the future of S_α, $\mathcal{D}$ is equivalent (in a suitable formal sense) to

$$\Sigma \cup \mathcal{D}_{ss} \cup \mathcal{D}_{ap} \cup \mathcal{D}_{una} \cup \mathcal{D}_{S_\alpha}$$

In other words, $\mathcal{D}_{S_\alpha}$ acts like the new initial database wrt all possible future evolutions of the theory following α.

To define progression, we first introduce an equivalence relation over structures. Let M and M' be structures (for our language) with the same domains for sorts *action* and *object*. Define $M' \sim_{S_\alpha} M$ iff the following two conditions hold:

1. M' and M interpret all predicate and function symbols which do not take any arguments of sort *state* identically.
2. M and M' agree on all fluents at S_α: For every predicate fluent F, and every variable assignment σ,

$$M', \sigma \models F(\vec{x}, do(\alpha, S_0)) \text{ iff } M, \sigma \models F(\vec{x}, do(\alpha, S_0)).$$

It is clear that $\sim_{S_\alpha}$ is an equivalence relation. If $M' \sim_{S_\alpha} M$, then M' agrees with M on S_α on fluents and state independent predicates and functions, but is free to vary its interpretation of everything else on all other states. In particular, they can interpret *Poss* and *do* differently. We have the following simple lemma.

Lemma 4.1 *If $M \sim_{S_\alpha} M'$, then for any formula φ in $\mathcal{L}^2_{S_\alpha}$, and any variable assignment σ, $M, \sigma \models \varphi$ iff $M', \sigma \models \varphi$.*

So we define

Definition 4.1 *A set of sentences $\mathcal{D}_{S_\alpha}$ in $\mathcal{L}^2_{S_\alpha}$ is a* progression *of the initial database $\mathcal{D}_{S_0}$ to S_α (wrt $\mathcal{D}$) iff for any structure M, M is a model of $\mathcal{D}_{S_\alpha}$ iff there is a model M' of $\mathcal{D}$ such that $M \sim_{S_\alpha} M'$.*

Notice that we define the new database only up to logical equivalence. We allow the new database to contain second-order sentences because, as we shall see later, first-order logic is not expressive enough for our purposes.

Proposition 4.1 *Let $\mathcal{D}_{S_\alpha}$ be a progression of the initial database to S_α. Then*

$$models(\mathcal{D}) \subseteq models(\Sigma \cup \mathcal{D}_{ss} \cup \mathcal{D}_{ap} \cup \mathcal{D}_{una} \cup \mathcal{D}_{S_\alpha}).$$

Proposition 4.2 *Let $\mathcal{D}_{S_\alpha}$ be a progression of the initial database to S_α. Then for every model M of*

$$\Sigma \cup \mathcal{D}_{ss} \cup \mathcal{D}_{ap} \cup \mathcal{D}_{una} \cup \mathcal{D}_{S_\alpha},$$

there exists a model M' of $\mathcal{D}$ such that:

1. *M' and M interpret all state independent predicate and function symbols identically.*
2. *For every variable assignment σ, and every predicate fluent F,*

$$M', \sigma \models S_\alpha \leq s \wedge F(\vec{x}, s) \text{ iff } M, \sigma \models S_\alpha \leq s \wedge F(\vec{x}, s).$$

Proof: Let M be a model of

$$\Sigma \cup \mathcal{D}_{ss} \cup \mathcal{D}_{ap} \cup \mathcal{D}_{una} \cup \mathcal{D}_{S_\alpha}.$$

Since M is a model of $\mathcal{D}_{S_\alpha}$, there is a model M' of

$$\Sigma \cup \mathcal{D}_{ss} \cup \mathcal{D}_{ap} \cup \mathcal{D}_{una} \cup \mathcal{D}_{S_0}$$

such that $M' \sim_{S_\alpha} M$. It can be easily seen that M' satisfies the desired properties. ■

From these two propositions, we conclude that $\mathcal{D}$ and $\Sigma \cup \mathcal{D}_{ss} \cup \mathcal{D}_{ap} \cup \mathcal{D}_{una} \cup \mathcal{D}_{S_\alpha}$ agree on all states $\geq S_\alpha$. So $\mathcal{D}_{S_\alpha}$ really does characterize the result of progressing the initial database in response to the action α. Furthermore, the following theorem says that the new database, when it exists, entails the same set of sentences in $\mathcal{L}^2_{S_\alpha}$ as $\mathcal{D}$:

Theorem 1 *Let $\mathcal{D}_{S_\alpha}$ be a progression of the initial database to S_α. For any sentence $\varphi \in \mathcal{L}^2_{S_\alpha}$, $\mathcal{D}_{S_\alpha} \models \varphi$ iff $\mathcal{D} \models \varphi$.*

Proof: If $\mathcal{D} \models \varphi$, then by Lemma 4.1, we have $\mathcal{D}_{S_\alpha} \models \varphi$. If $\mathcal{D}_{S_\alpha} \models \varphi$, then $\mathcal{D} \models \varphi$ by Proposition 4.1. ■

From this theorem, we see that if $\mathcal{D}_{S_\alpha}$ is a progression, then it is a *strongest post-condition* (cf. Pednault [9], Dijkstra and Scholten [3], and others) of the precondition $\mathcal{D}_{S_0}$ wrt the action α. A result by Pednault [9] shows that $\mathcal{D}_{S_\alpha}$ cannot in general be a finite set of first-order sentences in $\mathcal{L}_{S_\alpha}$. In the following, we shall extend this result, and show that $\mathcal{D}_{S_\alpha}$ cannot in general be a set of first-order sentences in $\mathcal{L}_{S_\alpha}$.

4.1 Progression Is Not Always First-Order Definable

At first glance, the fact that progression cannot always be done in first-order logic may seem obvious in light of Theorem 1, and the fact that $\mathcal{D}$ includes a second-order induction axiom. However, as we mentioned in section 3, for the purpose of progression, $\mathcal{D}$ is equivalent to $\mathcal{D} - \Sigma$, which is a finite set of first-order sentences.

We shall construct a basic action theory $\mathcal{D}$ and two structures M_1 and M_2 with the following properties:

1. $M_1 \models \mathcal{D}$.
2. M_1 and M_2 satisfy the exactly same set of sentences in $\mathcal{L}_{S_\alpha}$.
3. There is no model M' of $\mathcal{D}$ such that $M' \sim_{S_\alpha} M_2$.

It will then follow from our definition that for $\mathcal{D}$, the progression of the initial database to S_α cannot be in $\mathcal{L}_{S_\alpha}$. This is possible because for $M \sim_{S_\alpha} M'$ to hold, M and M' must be isomorphic with respect to sort *object*; but in number theory, there are nonstandard models that satisfy exactly the same set of first-order sentences as the standard model, and it is this property which we now use to show that progression is not always first-order definable.

We now proceed to construct a such basic action thery.

Consider the following theory $\mathcal{D}$ with a unary fluent F_1, and a binary fluent F_2, one action constant symbol A, one constant symbol 0, and one unary function symbol *succ*:

$\mathcal{D}_{S_0} = \emptyset$. $\mathcal{D}_{una} = \emptyset$.

$\mathcal{D}_{ap} = \{(\forall s).Poss(A, s) \equiv True\}$.

$\mathcal{D}_{ss}$ is the following pair of axioms:

$$Poss(a, s) \supset [F_1(do(a, s)) \equiv (\exists x)\neg F_2(x, s)],$$

$$\begin{aligned}
&Poss(a, s) \supset (\forall x).F_2(x, do(a, s)) \equiv \\
&\quad x = 0 \wedge F_2(0, s) \vee \\
&\quad F_2(x, s) \wedge (\exists y).x = succ(y) \wedge F_2(y, s) \vee \\
&\quad \neg F_2(x, s) \wedge x \neq 0 \wedge \\
&\qquad (\forall y)(x = succ(y) \supset \neg F_2(y, s)).
\end{aligned}$$

To understand the successor state axioms, think of the constant symbol 0 as the number 0, and the unary function *succ* as the successor function. F_1 simply keeps track of the truth value of F_2 in the previous state, and for $F_2(x, do(a, s))$ to be true, either $x = 0$ and $F_2(x, s)$, or both $F_2(x, s)$ and $F_2(predecessor(x), s)$ have the same truth values.

Consider a structure M such that:

1. M is a standard model of arithmetic with respect to sort *object*. Thus the domain for *object* in M is the set of nonnegative numbers, 0 is mapped to the number 0, and *succ* is mapped to the successor function.
2. $M \models F_1(do(A, S_0)) \wedge (\forall x).F_2(x, do(A, S_0))$.

Our first observation is that there cannot be a model M' of $\mathcal{D}$ such that $M \sim_{S_A} M'$. Suppose otherwise. Then M' also satisfies the above two properties 1 and 2. From $M' \models \mathcal{D}_{ss}$, and $M' \models F_1(do(A, S_0))$, we have $M' \models (\exists x)\neg F_2(x, S_0)$. Similarly, from $M' \models (\forall x).F_2(x, do(A, S_0))$, by the successor state axiom for F_2, we have $M' \models F_2(0, S_0) \wedge F_2(succ(0), S_0) \wedge \cdots$. Thus $M' \models (\forall x).F_2(x, S_0)$, a contradiction. Therefore there is not a model M' of $\mathcal{D}$ such that $M \sim_{S_\alpha} M'$.

We now show that there is a model M' of $\mathcal{D}$ such that for any sentence φ in $\mathcal{L}_{S_A}$, $M \models \varphi$ iff $M' \models \varphi$. By Skolem's theorem (cf. Kleene [5], page 326), there is a first-order structure M^* such that for any sentence φ in $\mathcal{L}_{S_A}$, $M \models \varphi$ iff $M^* \models \varphi$, and $(M, 0, succ)$ and $(M^*, 0, succ)$ are not isomorphic, i.e., M and M^* are not isomorphic on sort *object*. In particular, $M^* \models F_1(do(A, S_0)) \wedge (\forall x).F_2(x, do(A, S_0))$. Now revise M^* into a structure M' such that:

1. M' and M^* have the same domains for sorts *action* and *object*, and interpret state independent predicates and functions the same.
2. $M' \models (\forall a, s)Poss(a, s)$.
3. $M' \models \Sigma \cup \mathcal{D}_{una} \cup \mathcal{D}_{S_0}$.
4. For the truth values of the fluents on S_0: $M' \models F_1(S_0)$, and for the truth values of $F_2(x, S_0)$, we have that for any variable assignment σ:
   (a) If $\sigma(x)$ is a standard number, i.e., there is a $n \geq 0$ such that $M', \sigma \models x = succ^n(0)$, then $M', \sigma \models F_2(x, S_0)$.
   (b) If $\sigma(x)$ is a nonstandard number, i.e., there is no $n \geq 0$ such that $M', \sigma(x) \models x = succ^n(0)$, then $M', \sigma \models \neg F_2(x, S_0)$. Notice that since M^* and M are not isomorphic on sort *object* with respect to Peano arithmetic, there must be a nonstandard number in the domain of M^*, and thus in the domain of M'.
5. For the truth values of the fluents on $do(A, S_0)$: For any fluent F, and any variable assignment σ, $M', \sigma \models F(\vec{x}, do(A, S_0))$ iff $M^*, \sigma \models F(\vec{x}, do(A, S_0))$.

6. Inductively, for any variable assignment σ, if
$$M', \sigma \models do(A, S_0) < s,$$
then the truth values of the fluents on s will be determined according to the successor state axioms and the truth values of the fluents on $do(A, S_0)$; if
$$M', \sigma \models S_0 < s \wedge \neg do(A, S_0) < s,$$
then the truth values of the fluents on s will be determined according to the successor state axioms and the truth values of the fluents on S_0. This will define the truth values of the fluents on every state because $M' \models (\forall s).S_0 \leq s$, which follows from the fact that $M' \models (\forall a, s) Poss(a, s)$.

It is clear that $M' \sim_{S_A} M^*$. It follows that M' and M satisfy the same set of sentences in $\mathcal{L}_{S_A}$. We now show that M' satisfies the successor state axioms. By the construction of M', we only need to prove that it satisfies the successor state axioms instantiated to S_0 and action A, i.e.

$$M' \models Poss(A, S_0) \supset [F_1(do(A, S_0)) \equiv (\exists x) \neg F_2(x, S_0)],$$

and

$$\begin{aligned} M' \models Poss(A, S_0) \supset (\forall x).F_2(x, do(A, S_0)) \equiv \\ x = 0 \wedge F_2(0, S_0) \vee \\ F_2(x, S_0) \wedge (\exists y).x = succ(y) \wedge F_2(y, S_0) \vee \\ \neg F_2(x, S_0) \wedge x \neq 0 \wedge \\ (\forall y)(x = succ(y) \supset \neg F_2(y, S_0)). \end{aligned}$$

To show the first one, we need to prove that $M' \models (\exists x).\neg F_2(s, S_0)$. This follows from our construction of M' and the existence of nonstandard numbers in the domain of M'. To show the second one, we need to prove that

$$\begin{aligned} M' \models (\forall x).x = 0 \wedge F_2(0, S_0) \vee \\ F_2(x, S_0) \wedge (\exists y).x = succ(y) \wedge F_2(y, S_0) \vee \\ \neg F_2(x, S_0) \wedge x \neq 0 \wedge \\ (\forall y)(x = succ(y) \supset \neg F_2(y, S_0)). \end{aligned}$$

There are three cases:

1. If $x = 0$, then $F_2(0, S_0)$ follows from our construction.
2. If $x = succ^n(0)$ for some $n > 0$, then both $F_2(succ^n(0), S_0)$ and $F_2(succ^{n-1}(0), S_0)$ hold. Thus $F_2(x, S_0) \wedge (\exists y).x = succ(y) \wedge F_2(y, S_0)$ holds;
3. If x is a nonstandard number, then $F_2(x, S_0)$ does not hold. Furthermore, for any y such that $x = succ(y)$, y is also a nonstandard number, so $F_2(y, S_0)$ does not hold either. Thus $\neg F_2(x, S_0) \wedge x \neq 0 \wedge (\forall y)(x = succ(y) \supset \neg F_2(y, S_0))$ holds.

Therefore, M' satisfies the successor state axioms instantiated to S_0 and A. So $M' \models \mathcal{D}_{ss}$. This means that $M' \models \mathcal{D}$, and M' and M satisfy the same sentences in $\mathcal{L}_{S_A}$. Following the discussion at the beginning of the example, we see that the new database at S_A for $\mathcal{D}$ cannot be captured by a set of first-order sentences.

4.2 Progression Is Always Second-Order Definable

We now show that, by appealing to second-order logic, progression always exists. We shall first introduce some notation.

Given a finite set $\mathcal{D}_{ss}$ of successor state axioms, we define the *instantiation* of $\mathcal{D}_{ss}$ on an action term at and a state term st, written $\mathcal{D}_{ss}[at, st]$, to be the sentence:

$$\bigwedge_{F \text{ is a fluent}} \begin{array}{l} Poss(at, st) \supset \\ (\forall \vec{x}).F(\vec{x}, do(at, st)) \equiv \Phi_F(\vec{x}, at, st), \end{array}$$

where

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

is the successor state axiom for F in $\mathcal{D}_{ss}$.

Given a formula φ in $\mathcal{L}^2$, the *lifting* of φ on the state st, written $\varphi \upharpoonright st$, is the result of replacing every fluent atom of the form $F(t_1, ..., t_n, st)$ by a new predicate variable $p(t_1, ..., t_n)$ of arity $object^n$. For instance,

$$enrolled(John, C200, S_0) \wedge enrolled(John, C100, S_0) \upharpoonright S_0$$

is $p(John, C200) \wedge p(John, C100)$.[5]

Lemma 4.2 *The following are some simple properties of lifting:*

1. *If φ is a sentence that does not mention st, then $\varphi \upharpoonright st$ is φ.*
2. *If φ is a sentence in $\mathcal{L}^2_{st}$, then $\varphi \upharpoonright st$ is a state independent sentence.*
3. *If φ does not contain quantifiers over states, then $\varphi \models \varphi \upharpoonright st$.*

Now we can state the main theorem of this section:

Theorem 2 *Let $\mathcal{D}_{S_\alpha}$ be the union of $\mathcal{D}_{una}$ together with the sentence:*

$$(\exists p_1, ..., p_k)\{ \bigwedge_{\varphi \in \mathcal{D}_{S_0}} \varphi \wedge \mathcal{D}_{ss}[\alpha, S_0](Poss/\Psi_\alpha)\} \upharpoonright S_0,$$

where

1. *$p_1, ..., p_k$ are the new predicate variables introduced during the lifting.*

[5] Lifting as we have defined it does not generally preserve logical equivalence. For instance, $[(\forall s).F(s)] \upharpoonright S_0$ is $(\forall s).F(s)$, but the logically equivalent $[F(S_0) \wedge (\forall s).F(s)] \upharpoonright S_0$ is $p \wedge (\forall s).F(s)$. Fortunately, we shall only be lifting those sentences that do preserve logical equivalence.

2. Ψ_α is a sentence in $\mathcal{L}_{S_0}$ such that

$$Poss(\alpha, S_0) \equiv \Psi_\alpha$$

is an instance of the the axiom in $\mathcal{D}_{ap}$ corresponding to the action of α.

3. $\mathcal{D}_{ss}[\alpha, S_0](Poss/\Psi_\alpha)$ is the result of replacing $Poss(\alpha, S_0)$ by Ψ_α in $\mathcal{D}_{ss}[\alpha, S_0]$.

Then $\mathcal{D}_{S_\alpha}$ is a progression of $\mathcal{D}_{S_0}$ to S_α wrt $\mathcal{D}$:

Proof: First, it is clear that the sentences in $\mathcal{D}_{S_\alpha}$ are in $\mathcal{L}^2_{S_\alpha}$.

Let M be a structure. We need to show that $M \models \mathcal{D}_{S_\alpha}$ iff there is a model M' of $\mathcal{D}$ such that $M \sim_{S_\alpha} M'$.

Suppose that there is a model M' of $\mathcal{D}$ such that $M \sim_{S_\alpha} M'$. By Lemma 4.2, $\mathcal{D} \models \mathcal{D}_{S_\alpha}$, thus $M' \models \mathcal{D}_{S_\alpha}$. Therefore by Lemma 4.1, $M \models \mathcal{D}_{S_\alpha}$.

Now suppose that $M \models \mathcal{D}_{S_\alpha}$. Then there is a variable assignment σ such that

$$M, \sigma \models \bigwedge_{\varphi \in \mathcal{D}_{S_0}} \varphi \wedge \mathcal{D}_{ss}[\alpha, S_0](Poss/\Psi_\alpha){\uparrow} S_0.$$

Now construct a structure M' such that

1. M and M' have the same universe, and interpret all state independent function and predicate symbols identically.
2. For every fluent F, if $F(\vec{x}, S_0)$ is lifted in $\mathcal{D}_{S_\alpha}$ as p, then
$$M', \sigma \models F(\vec{x}, S_0) \text{ iff } M, \sigma \models p(\vec{x}).$$
3. $M' \models \mathcal{D}_{ss} \cup \mathcal{D}_{ap}$.
4. If $M' \models \neg\Psi_\alpha$, then for any fluent F, and any variable assignment σ',
$$M', \sigma' \models F(\vec{x}, S_\alpha) \text{ iff } M, \sigma' \models F(\vec{x}, S_\alpha).$$

It is clear that such a M' exists. We claim that $M \sim_{S_\alpha} M'$. There are two cases:

1. If $M' \models \neg\Psi_\alpha$, then it follows from our construction that for any fluent F, and any variable assignment σ',

$$M', \sigma' \models F(\vec{x}, S_\alpha) \text{ iff } M, \sigma' \models F(\vec{x}, S_\alpha).$$

2. If $M' \models \Psi_\alpha$, then since $M' \models \mathcal{D}_{ap}$, and $\mathcal{D}_{ap} \models Poss(\alpha, S_0) \equiv \Psi_\alpha$, therefore $M' \models Poss(\alpha, S_0)$. But $M' \models \mathcal{D}_{ss}$. Thus for any fluent F, and any variable assignment σ',

$$M', \sigma' \models F(\vec{x}, S_\alpha) \text{ iff } M', \sigma' \models \Phi_F(\vec{x}, \alpha, S_0), \quad (1)$$

where Φ_F is as in the successor state axiom for F in $\mathcal{D}_{ss}$. Now since $M' \models \Psi_\alpha$, by our construction of M', we have that $M, \sigma \models \Psi_\alpha{\uparrow} S_0$. But

$$M, \sigma \models \mathcal{D}_{ss}[\alpha, S_0](Poss/\Psi_\alpha){\uparrow} S_0.$$

Therefore for any fluent F, and any variable assignment σ' such that $\sigma'(p) = \sigma(p)$ for any predicate variable p,

$$M, \sigma' \models F(\vec{x}, S_\alpha) \text{ iff } M', \sigma' \models \Phi_F(\vec{x}, \alpha, S_0){\uparrow} S_0. \quad (2)$$

But for any variable assignment σ' such that $\sigma'(p) = \sigma(p)$ for any predicate variable p, since $\Phi_F(\vec{x}, \alpha, S_0)$ is in $\mathcal{L}_{S_0}$, by our construction of M',

$$M, \sigma' \models \Phi_F(\vec{x}, \alpha, S_0){\uparrow} S_0 \text{ iff } M', \sigma' \models \Phi_F(\vec{x}, \alpha, S_0),$$

Therefore from (1) and (2), we see that for any fluent F, and any variable assignment σ',

$$M', \sigma' \models F(\vec{x}, S_\alpha) \text{ iff } M, \sigma' \models F(\vec{x}, S_\alpha).$$

It follows then that $M \sim_{S_\alpha} M'$. By the construction of M' and the fact that $M \models \mathcal{D}_{una}$, we have that $M' \models \mathcal{D}_{ss} \cup \mathcal{D}_{ap} \cup \mathcal{D}_{una}$. Thus from Proposition 3.1, there is a model M'' of $\mathcal{D}$ such that $M' \sim_{S_\alpha} M''$. Then by the transitivity of $\sim_{S_\alpha}$, we have that $M \sim_{S_\alpha} M''$. This concludes the proof that $\mathcal{D}_{S_\alpha}$ as defined is progressed database. ∎

It is clear that the theorem still holds when the initial database $\mathcal{D}_{S_0}$ is a finite set of second-order sentences in $\mathcal{L}^2_{S_0}$. Therefore, at least in principle, the theorem can be repeatedly applied to progress the initial database in response to a sequence of actions.

The new database $\mathcal{D}_{S_\alpha}$ as defined in the theorem can be unwieldy. However, it can often be simplified by using the unique names axioms in $\mathcal{D}_{una}$, as we shall see in the following example.

Example 4.1 Consider our educational database. The instantiation of the successor state axioms on $drop(Sue, C100)$ and S_0, $\mathcal{D}_{ss}[drop(Sue, C100), S_0]$, is the conjunction of the following two sentences, where $\alpha = drop(Sue, C100)$ and $S_\alpha = do(\alpha, S_0)$:

$$\begin{aligned}&Poss(\alpha, S_0) \supset enrolled(st, c, S_\alpha) \equiv \\ &\quad \alpha = register(st, c) \vee \\ &\quad enrolled(st, c, S_0) \wedge \alpha \neq drop(st, c),\end{aligned}$$

$$\begin{aligned}&Poss(\alpha, S_0) \supset grade(st, c, g, S_\alpha) \equiv \\ &\quad \alpha = change(st, c) \vee \\ &\quad grade(st, c, g, s) \wedge (\forall g')\alpha \neq change(st, c, g').\end{aligned}$$

By unique names axioms, these two sentences can be simplified to

$$\begin{aligned}&Poss(\alpha, S_0) \supset enrolled(st, c, S_\alpha) \equiv \\ &\quad enrolled(st, c, S_0) \wedge (Sue \neq st \vee C100 \neq c),\end{aligned}$$

$$Poss(\alpha, S_0) \supset grade(st, c, g, S_\alpha) \equiv grade(st, c, g, s).$$

By $\mathcal{D}_{ap}$,

$$Poss(\alpha, S_0) \equiv enrolled(Sue, C100, S_0).$$

Thus $\mathcal{D}_{ss}[\alpha, S_0](Poss/\Psi_\alpha)$ is the conjunction of the following two sentences:

$$enrolled(Sue, C100, S_0) \supset enrolled(st, c, S_\alpha) \equiv enrolled(st, c, S_0) \wedge (Sue \neq st \vee C100 \neq c),$$

$$enrolled(Sue, C100, S_0) \supset grade(st, c, g, S_\alpha) \equiv grade(st, c, g, s).$$

Thus $(\exists p_1, p_2)[\bigwedge_{\varphi \in \mathcal{D}_{S_0}} \varphi \wedge \mathcal{D}_{ss}[\alpha, S_0](Poss/\Psi_\alpha)] \uparrow S_0$ is

$$\begin{aligned}&(\exists p_1, p_2).John \neq Sue \neq C100 \neq C200 \wedge\\&\quad p_1(John, C100) \vee p_1(John, C200) \wedge\\&\quad p_1(Sue, C100) \wedge prerequ(C100, C200) \wedge\\&\quad p_1(Sue, C100) \supset enrolled(st, c, S_\alpha) \equiv\\&\qquad p_1(st, c) \wedge (Sue \neq st \vee C100 \neq c) \wedge\\&\quad p_1(Sue, C100) \supset grade(st, c, g, S_\alpha) \equiv\\&\qquad p_2(st, c, g).\end{aligned}$$

This is equivalent to

$$\begin{aligned}&John \neq Sue \neq C100 \neq C200 \wedge\\&prerequ(C100, C200) \wedge\\&(\exists p_1, p_2).p_1(John, C100) \vee p_1(John, C200) \wedge\\&\quad p_1(Sue, C100) \wedge\\&\quad enrolled(st, c, S_\alpha) \equiv\\&\qquad p_1(st, c) \wedge (Sue \neq st \vee C100 \neq c) \wedge\\&\quad grade(st, c, g, S_\alpha) \equiv p_2(st, c, g),\end{aligned}$$

which is equivalent to

$$\begin{aligned}&John \neq Sue \neq C100 \neq C200 \wedge\\&prerequ(C100, C200) \wedge\\&(\exists p_1).p_1(John, C100) \vee p_1(John, C200) \wedge\\&\quad p_1(Sue, C100) \wedge\\&\quad enrolled(st, c, S_\alpha) \equiv\\&\qquad p_1(st, c) \wedge (Sue \neq st \vee C100 \neq c),\end{aligned}$$

which is equivalent to

$$\begin{aligned}&John \neq Sue \neq C100 \neq C200 \wedge\\&prerequ(C100, C200) \wedge\\&enrolled(John, C100, S_\alpha) \vee\\&\qquad enrolled(John, C200, S_\alpha) \wedge\\&\neg enrolled(Sue, C100, S_\alpha) \wedge\\&(\exists p_1).enrolled(st, c, S_\alpha) \equiv p_1(st, c).\end{aligned}$$

Finally, we have a first-order representation for $\mathcal{D}_{S_\alpha}$, which is $\mathcal{D}_{una}$ together with the following sentences:

$$John \neq Sue \neq C100 \neq C200,$$

$$prerequ(C100, C200),$$

$$enrolled(John, C100, S_\alpha) \vee enrolled(John, C200, S_\alpha),$$

$$\neg enrolled(Sue, C100, S_\alpha).$$

■

To summarize, we have shown that in general, progression is definable only in second-order logic. However, there are some interesting special cases for which progression can be done in first-order logic. We shall give two such special cases.

5 PROGRESSION WITH RELATIVELY COMPLETE INITIAL DATABASES

We say $\mathcal{D}_{S_0}$ is *relatively complete (wrt state independent propositions)* if it is a set of state independent sentences together with a set of sentences, one for each fluent F, of the form:

$$(\forall \vec{x}).F(\vec{x}, S_0) \equiv \Pi_F(\vec{x}),$$

where $\Pi_F(\vec{x})$ is a state independent formula whose free variables are among $\vec{x}$. Clearly, for relatively complete $\mathcal{D}_{S_0}$, if it is complete about the state independent sentences: For any state independent sentence Π,

$$\text{either } \mathcal{D}_{S_0} \models \Pi \text{ or } \mathcal{D}_{S_0} \models \neg\Pi,$$

then it is also complete about S_0: For any sentence φ in $\mathcal{L}_{S_0}$,

$$\text{either } \mathcal{D}_{S_0} \models \varphi \text{ or } \mathcal{D}_{S_0} \models \neg\varphi.$$

Theorem 3 *Let $\mathcal{D}$ be an action theory with a relatively complete initial database $\mathcal{D}_{S_0}$, and let α be a ground action term such that $\mathcal{D} \models Poss(\alpha, S_0)$. Then the following set:*

$$\begin{aligned}&\mathcal{D}_{una} \cup \{\varphi \mid \varphi \in \mathcal{D}_{S_0} \textit{ is state independent}\} \cup\\&\{(\forall \vec{x}).F(\vec{x}, do(\alpha, S_0)) \equiv \Phi_F(\vec{x}, \alpha, S_0)[S_0] \mid\\&\qquad F \textit{ is a fluent}\}\end{aligned}$$

is a progression of $\mathcal{D}_{S_0}$ to S_α, where

1. *$\Phi_F(\vec{x}, \alpha, S_0)$ is as in the successor state axiom for F in $\mathcal{D}_{ss}$;*
2. *$\Phi_F(\vec{x}, \alpha, S_0)[S_0]$ is the result of replacing, in $\Phi_F(\vec{x}, \alpha, S_0)$, every occurrence of $F'(\vec{t}, S_0)$ by $\Pi_{F'}(\vec{t})$, where $\Pi_{F'}$ is as in the corresponding axiom for F' in $\mathcal{D}_{S_0}$, and this replacement is performed for every fluent F' mentioned in $\Phi_F(\vec{x}, \alpha, S_0)$.*

Proof: Denote the set of the sentences of the theorem by $\mathcal{S}$. Clearly, $\mathcal{S}$ is a set of first-order sentences in $\mathcal{L}_{S_\alpha}$. It is easy to see that $\mathcal{S} \models \mathcal{D}_{S_\alpha}$. Conversely, it is clear that $\mathcal{D} \models \mathcal{S}$. Thus by Theorem 1, $\mathcal{D}_{S_\alpha} \models \mathcal{S}$. ■

Clearly, the progressed database at S_α as given by the theorem is also relatively complete. Thus the theorem can be repeatedly applied to progress a relatively complete initial database in response to a sequence of executable actions. Notice that the new database will include $\mathcal{D}_{una}$ and the state independent axioms in $\mathcal{D}_{S_0}$; therefore we can use these axioms to simplify $\Phi_F(\vec{x}, \alpha, S_0)[S_0]$.

Example 5.1 Consider again our educational database example. Suppose now that the initial database $\mathcal{D}_{S_0}$ consists of the following axioms:

$$John \neq Sue \neq C100 \neq C200,$$

$$better(70, 50),$$

$$prerequ(C100, C200),$$

$$enrolled(st, c, S_0) \equiv \\ (st = John \wedge c = C100) \vee (st = Sue \wedge c = C200),$$

$$grade(st, c, g, S_0) \equiv \\ st = Sue \wedge c = C100 \wedge g = 70.$$

Clearly $\mathcal{D}_{S_0}$ is relatively complete, and $\mathcal{D} \models Poss(\alpha, S_0)$, where $\alpha = drop(John, C100)$. From the axiom for *enrolled* in $\mathcal{D}_{S_0}$, we see that $\Pi_{enrolled}(st, c)$ is the formula:

$$(st = John \wedge c = C100) \vee (st = Sue \wedge c = C200).$$

Now from the successor state axiom for *enrolled* in Example 3.1, we see that $\Phi_{enrolled}(st, c, a, s)$, the condition under which $enrolled(st, c, do(a, s))$ will be true, is the formula:

$$a = register(st, c) \vee (enrolled(st, c, s) \wedge a \neq drop(st, c)).$$

Therefore $\Phi_{enrolled}(st, c, \alpha, S_0)[S_0]$ is the formula:

$$drop(John, C100) = register(st, c) \vee \\ \{[(st = John \wedge c = C100) \vee (st = Sue \wedge c = C200)] \wedge \\ drop(John, C100) \neq drop(st, c))\}.$$

By the unique names axioms in $\mathcal{D}_{una}$, this can be simplified to

$$(st = John \wedge c = C100) \vee (st = Sue \wedge c = C200) \wedge \\ (John \neq st \vee C100 \neq c).$$

By the unique names axioms in $\mathcal{D}_{S_0}$, this can be further simplified to

$$st = Sue \wedge c = C200.$$

Therefore we obtain the following axiom about $do(\alpha, S_0)$:

$$enrolled(st, c, do(\alpha, S_0)) \equiv st = Sue \wedge c = C200.$$

Similarly, we have:

$$grade(st, c, g, do(\alpha, S_0)) \equiv st = Sue \wedge c = C100 \wedge g = 70.$$

Therefore a progression to $do(drop(John, C100), S_0)$ is $\mathcal{D}_{una}$ together with the following sentences:

$$John \neq Sue \neq C100 \neq C200,$$
$$better(70, 50),$$
$$prerequ(C100, C200),$$
$$enrolled(st, c, do(\alpha, S_0)) \equiv st = Sue \wedge c = C200,$$
$$grade(st, c, g, do(\alpha, S_0)) \equiv st = Sue \wedge c = C100 \wedge g = 70.$$

■

6 PROGRESSION IN THE CONTEXT FREE CASE

In this section we consider progression wrt context-free action theories. A successor state axiom for F is *context free* iff it has the form:

$$Poss(a, s) \supset F(\vec{x}, do(a, s)) \equiv \\ (\exists \vec{u})(a = A_1(\vec{\xi_1}, \vec{u}) \wedge E_1) \vee \cdots \vee \\ (\exists \vec{v})(a = A_m(\vec{\xi_m}, \vec{v}) \wedge E_m) \ \vee \\ F(\vec{x}, s) \wedge \neg(\exists \vec{w})(a = B_1(\vec{\chi_1}, \vec{w}) \wedge E_{m+1}) \wedge \cdots \wedge \\ \neg(\exists \vec{r})(a = B_n(\vec{\chi_n}, \vec{r}) \wedge E_{m+n}),$$

where $\vec{\xi_i}$ and $\vec{\chi_j}$ denote sequences of all, or just some (including none) of the $\vec{x}$, the A's and B's are actions, and $E_1, \ldots, E_{m+n}$ are propositional formulas constructed from equality literals over the domain objects, i.e., they are quantifier free, and do not mention terms of sort *state* and *action*. The successor state axioms in our educational database are all context free. So are the following successor state axioms:

$$Poss(a, s) \supset holding(x, do(a, s)) \equiv a = pickup(x) \ \vee \\ holding(x, s) \wedge a \neq drop(x) \wedge \neg(\exists u) a = put(x, u).$$

$$Poss(a, s) \supset on(x, y, do(a, s)) \equiv a = move(x, y) \ \vee \\ on(x, y, s) \wedge \neg(\exists z)(a = move(x, z) \wedge z \neq y).$$

The following successor state axiom is not context-free:

$$Poss(a, s) \supset dead(x, do(a, s)) \equiv \\ (\exists y).a = explode\_bomb\_at(y) \wedge close(x, y, s) \ \vee \\ dead(x, s).$$

Given any action terms $A_1(\vec{t_1})$ and $A_2(\vec{t_2})$, by the unique names axioms, the equality $A_1(\vec{t_1}) = A_2(\vec{t_2})$ is either equivalent to *false* or, when A_1 and A_2 are the same, equivalent to $\vec{t_1} = \vec{t_2}$.[6] Thus, given any action term $A(\vec{t})$, the instantiation of a context-free successor state axiom on $A(\vec{t})$ is equivalent to

$$Poss(A(\vec{t}), s) \supset \\ F(\vec{x}, do(A(\vec{t}), s)) \equiv [E_F \vee (F(\vec{x}, s) \wedge \neg E_{\neg F})],$$

where E_F and $E_{\neg F}$ are propositional formulas constructed from equality literals over the domain objects. This is logically equivalent to

$$Poss(A(\vec{t}), s) \supset \\ [E_F \vee (F(\vec{x}, s) \wedge \neg E_{\neg F})] \supset \\ F(\vec{x}, do(A(\vec{t}), s)), \qquad (3)$$

[6] $\vec{x} = \vec{y}$ is an abbreviation for $x_1 = y_1 \wedge \cdots \wedge x_n = y_n$. Notice that when both $\vec{x}$ and $\vec{y}$ are the empty sequence, $\vec{x} = \vec{y}$ is logically equivalent to *true*. It is equivalent to *false* when $\vec{x}$ and $\vec{y}$ have different length.

$$Poss(A(\vec{t}), s) \supset$$
$$[\neg E_F \wedge (\neg F(\vec{x}, s) \vee E_{\neg F})] \supset$$
$$\neg F(\vec{x}, do(A(\vec{t}), s)). \quad (4)$$

For instance, by the above successor state axiom for *holding*, we have

$$Poss(drop(x), s) \supset holding(y, do(drop(x), s)) \equiv holding(y, s) \wedge y \neq x.$$

Here $E_{holding}$ is $false$, and $E_{\neg holding}$ is $x = y$.

Now assume that:

1. $\mathcal{D}_{S_0}$ is a set of state independent sentences, and sentences of the form
$$E \supset \pm F(x_1, \ldots, x_n, S_0), \quad (5)$$
where E is a propositional formula constructed from equality literals over the domain objects. For example,
$$ontable(x, S_0),$$
$$x \neq A \supset \neg ontable(x, S_0),$$
$$x = A \wedge y = B \supset on(x, y, S_0),$$
are all of this form.
2. $\mathcal{D}_{S_0}$ is *coherent* in the sense that for every fluent F, whenever $(\forall \vec{x}).E_1 \supset F(\vec{x}, S_0)$ and $(\forall \vec{x}).E_2 \supset \neg F(\vec{x}, S_0)$ are in $\mathcal{D}_{S_0}$, then
$$\{\varphi | \varphi \in \mathcal{D}_{S_0} \text{ is state independent}\} \models (\forall \vec{x}).\neg(E_1 \wedge E_2).$$
This means that $\mathcal{D}_{S_0}$ cannot use axioms of the form (5) to encode state independent sentences: For any state independent sentence ϕ, $\mathcal{D}_{S_0} \models \phi$ iff
$$\{\varphi \mid \varphi \in \mathcal{D}_{S_0} \text{ is state independent}\} \models \phi.$$
3. $\mathcal{D}_{ss}$ is a set of context-free successor state axioms.
4. α is a ground action term, say $A(\vec{t})$.
5. α is possible initially: $\mathcal{D} \models Poss(\alpha, S_0)$.

For example, our educational database in Example 3.1 with the following initial database:

$$Sue \neq John \neq C100 \neq C200 \neq 50,$$
$$st = Sue \wedge c = C100 \supset enrolled(st, c, S_0),$$
$$st = Sue \wedge c = C200 \supset \neg enrolled(st, c, S_0),$$
$$st = Sue \wedge c = C100 \wedge g = 50 \supset grade(st, c, g, S_0),$$

satisfies the above conditions for $\alpha = drop(Sue, C100)$.

To compute $\mathcal{D}_{S_\alpha}$, use Theorem 1 to construct a set $\mathcal{S}$, initially empty, of sentences as follows:

1. If $\varphi \in \mathcal{D}_{S_0}$ is state independent, then $\varphi \in \mathcal{S}$.
2. For any fluent F, by (3) and (4), the coherence assumption, and the assumption that $\mathcal{D} \models Poss(\alpha, S_0)$, add to $\mathcal{S}$ the sentences
$$E_F \supset F(\vec{x}, do(\alpha, S_0)),$$
$$E_{\neg F} \supset \neg F(\vec{x}, do(\alpha, S_0)).$$
3. For any fluent F, if $(\forall \vec{x}).E \supset F(\vec{x}, S_0)$ is in $\mathcal{D}_{S_0}$, then, by (3) and the assumption that $\mathcal{D} \models Poss(\alpha, S_0)$, add to $\mathcal{S}$ the sentence
$$E \wedge \neg E_{\neg F} \supset F(\vec{x}, do(\alpha, S_0)).$$
4. For any fluent F, if $(\forall \vec{x}).E \supset \neg F(\vec{x}, S_0)$ is in $\mathcal{D}_{S_0}$, then, by (4) and the assumption that $\mathcal{D} \models Poss(\alpha, S_0)$, add to $\mathcal{S}$ the sentence
$$\neg E_F \wedge E \supset \neg F(\vec{x}, do(\alpha, S_0)).$$

For example, consider again our educational database with the above initial database, and

$$\alpha = drop(Sue, C100),$$

we have

$$Poss(\alpha, s) \supset enrolled(st, c, do(\alpha, s)) \equiv enrolled(st, c, s) \wedge \neg(st = Sue \wedge c = C100),$$

$$Poss(\alpha, s) \supset grade(st, c, g, do(\alpha, s)) \equiv grade(st, c, g, s).$$

Thus $E_{enrolled}$ is $False$, $E_{\neg enrolled}$ is

$$st = Sue \wedge c = C100,$$

and E_{grade} and $E_{\neg grade}$ are both $False$. Then the above procedure will give us the following set $\mathcal{S}$:

$$John \neq Sue \neq C100 \neq C200 \neq 50,$$
$$(st = Sue \wedge c = C100) \supset \neg enrolled(st, c, S_\alpha),$$
$$(st = Sue \wedge c = C200) \supset \neg enrolled(st, c, S_\alpha),$$
$$st = Sue \wedge c = C100 \wedge g = 50 \supset grade(st, c, g, S_\alpha).$$

As we show in the following theorem, together with $\mathcal{D}_{una}$, this is a progression of $\mathcal{D}_{S_0}$ to S_α,

Theorem 4 *Under the afore-mentioned assumptions, $\mathcal{S} \cup \mathcal{D}_{una}$ is a progression of $\mathcal{D}_{S_0}$ to S_α.*

Proof: It is clear that $\mathcal{D} \models \mathcal{S} \cup \mathcal{D}_{una}$, and $\mathcal{S}$ is a set of sentences in $\mathcal{L}_{S_\alpha}$. Therefore by Theorem 1, $\mathcal{D}_{S_\alpha} \models \mathcal{S} \cup \mathcal{D}_{una}$. To prove the converse, we show that for any model M of $\mathcal{S} \cup \mathcal{D}_{una}$, there is a model M' of $\mathcal{D}$ such that $M \sim_{S_\alpha} M'$. Suppose now that M is a model of $\mathcal{S} \cup \mathcal{D}_{una}$. We construct M' as follows:

1. M' and M have the same domains for sorts *action* and *object*, and interpret all state independent predicates and functions the same.
2. For each fluent F, M' interprets it on S_0 as follows:

(a) For every variable assignment σ, if $(\forall \vec{x}).E \supset F(\vec{x}, S_0)$ is in $\mathcal{D}_{S_0}$, and $M, \sigma \models E$ (thus $M', \sigma \models E$ as well), then $M', \sigma \models F(\vec{x}, S_0)$.

(b) Similarly, for every variable assignment, if $(\forall \vec{x}).E \supset \neg F(\vec{x}, S_0)$ is in $\mathcal{D}_{S_0}$, and $M, \sigma \models E$ (thus $M', \sigma \models E$ as well), then $M', \sigma \models \neg F(\vec{x}, S_0)$.

(c) For every variable assignment σ, if $F(\vec{x}, S_0)$ has not been assigned a truth value by one of the above two steps, then $M', \sigma \models F(\vec{x}, S_0)$ iff $M, \sigma \models F(\vec{x}, do(\alpha, S_0))$.

Notice that by the coherence assumption for $\mathcal{D}_{S_0}$, our construction is well-defined.

3. M' interprets $Poss$ according to $\mathcal{D}_{ap}$, and interprets the truth values of the fluents on reachable states according to $\mathcal{D}_{ss}$.

4. $M' \models \Sigma$. This can be done according to Proposition 3.1.

Clearly $M' \models \mathcal{D}$. We show now that $M \sim_{S_\alpha} M'$. For any fluent F, suppose the successor state axiom for it is

$$Poss(\alpha, s) \supset F(\vec{x}, do(\alpha, s)) \equiv E_F \vee (F(\vec{x}, s) \wedge \neg E_{\neg F}).$$

Given a variable assignment σ, suppose $M', \sigma \models F(\vec{x}, do(\alpha, S_0))$. Since $\mathcal{D} \models Poss(\alpha, S_0)$, by the above successor state axiom, there are two cases:

1. $M', \sigma \models E_F$. This implies $M, \sigma \models E_F$. Now since $E_F \supset F(\vec{x}, do(\alpha, S_0)) \in \mathcal{S}$, and M is a model of $\mathcal{S}$, thus $M, \sigma \models F(\vec{x}, do(\alpha, S_0))$ as well.

2. $M', \sigma \models \neg E_F \wedge F(\vec{x}, S_0) \wedge \neg E_{\neg F}$. From $M', \sigma \models F(\vec{x}, S_0)$, by our construction, either $M, \sigma \models F(\vec{x}, do(\alpha, S_0))$, or there is a sentence $E \supset F(\vec{x}, S_0)$ in $\mathcal{D}_{S_0}$ such that $M, \sigma \models E$. Suppose it is the latter. Then by our construction of $\mathcal{S}$, it contains $E \wedge \neg E_{\neg F} \supset F(\vec{x}, do(\alpha, S_0))$. Thus $M, \sigma \models F(\vec{x}, do(\alpha, S_0))$ as well.

Similarly, if $M', \sigma \models \neg F(\vec{x}, do(\alpha, S_0))$, then $M, \sigma \models \neg F(\vec{x}, do(\alpha, S_0))$ as well. Therefore $M \sim_{S_\alpha} M'$. ■

We have some remarks:

1. The new database $\mathcal{S}$ has the same form as $\mathcal{D}_{S_0}$, so this process can be iterated.
2. The generation of $\mathcal{S}$ is very fast, and the size of $\mathcal{S}$ is bounded by the sum of the size of $\mathcal{D}_{S_0}$ and the twice the number of fluents.
3. The E's in context free successor state axioms can be any state independent formulas. Thus a limited context dependency can be handled.

We emphasize that the results of this section depend on the fact that the initial database has a certain specific form. In fact, a result by Pednault [9] shows that for context-free actions and arbitrary $\mathcal{D}_{S_0}$, progression is not always guaranteed to yield finite first-order theories.

7 SUMMARY

1. We have argued the need for progressing a database.

2. We have defined a formal notion of progression, and showed that in general, to capture it we need second-order logic.

3. We have studied two special cases for which progression is first order definable, and which can be done efficiently.

4. Although we don't discuss them here, there are other cases for which progression can be done in first order logic. One such case concerns actions with finitary effects, i.e. for any fluent, the action changes the truth values of the fluent at only a finite number of instances.

5. The complexity of progression depends on both the form of the initial database, and the form of the action theory. A relatively complete initial database can be progressed efficiently wrt any successor state axioms. On the other hand, even for context free successor state axioms, progression is not guaranteed to yield finite first-order theories.

6. In a companion paper (Lin and Reiter [7]) we explore the consequences of our results on progression for the semantics of STRIPS-like systems. Ever since STRIPS was first introduced (Fikes and Nilsson [4]), its logical semantics has been problematic. There have been many proposals in the literature (e.g. Lifschitz [6], Pednault [11], Bacchus and Yang [2]). These all have in common a reliance on meta-theoretic operations on logical theories in order to capture the add and delete lists of STRIPS operators, but it has never been clear exactly what these operations correspond to declaratively, especially when they are applied to logically incomplete theories. In the companion to this paper, we provide a semantics for STRIPS-like systems in terms of basic theories of actions in the situation calculus. On our view, STRIPS is a *mechanism* for computing the progression of an initial situation calculus database under the effects of an action. We illustrate this idea by specifying two different versions of STRIPS in the situation calculus as well as a generalization of STRIPS that appeals to relational database theory.

Acknowledgements

For their generous advice and feedback, we wish to thank the other members of the University of Toronto Cognitive Robotics Group: Yves Lespérance, Hector Levesque, Bill Millar, Daniel Marcu, and Richard Scherl. This research was funded by the Government of Canada National Sciences and Engineering Research Council, and the Institute for Robotics and Intelligent Systems.

References

[1] S. Abiteboul. Updates, a new frontier. In *Second International Conference on Database Theory*, pages 1–18. Springer, 1988.

[2] F. Bacchus and Q. Yang. Downward refinement and the efficiency of hierarchical problem solving. *Artificial Intelligence*. To appear.

[3] E. W. Dijkstra and C. S. Scholten. *Predicate Calculus and Program Semantics*. Springer-Verlag, New York, 1990.

[4] R. E. Fikes and N. J. Nilsson. STRIPS: A new approach to theorem proving in problem solving. *Artificial Intelligence*, 2:189–208, 1971.

[5] S. C. Kleene. *Mathematical Logic*. John Wiley & Sons, Inc., 1967.

[6] V. Lifschitz. On the semantics of STRIPS. In *Reasoning about Actions and Plans: Proceedings of the 1986 Workshop*, pages 1–9. Morgan Kauffmann Publishers, Inc., 1986. June 30–July 2, Timberline, Oregon.

[7] F. Lin and R. Reiter. How to progress a database II: The STRIPS connection. 1994. Submitted.

[8] F. Lin and R. Reiter. State constraints revisited. *Journal of Logic and Computation Special Issue on Actions and Processes*, 1994. To appear.

[9] E. P. Pednault. *Toward a Mathematical Theory of Plan Synthesis*. PhD thesis, Department of Electrical Engineering, Stanford University, Stanford, CA, 1986.

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

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

[12] R. Reiter. The frame problem in the situation calculus: a simple solution (sometimes) and a completeness result for goal regression. In V. Lifschitz, editor, *Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy*, pages 418–420. Academic Press, San Diego, CA, 1991.

[13] R. Reiter. On specifying database updates. Technical report, Department of Computer Science, University of Toronto, 1992. KRR-TR-92-3.

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

[15] S. J. Rosenschein. Plan synthesis: A logical perspective. In *Proceedings of IJCAI 7*, pages 331–337, 1981.

[16] R. Scherl and H. Levesque. The frame problem and knowledge-producing actions. In *Proceedings of the Eleventh National Conference on Artificial Intelligence (AAAI–93)*, 1993.

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