How to Progress a Database II: The STRIPS Connection

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

Abstract

Ever since STRIPS was first introduced (Fikes and Nilsson [3]), its logical semantics has been problematic. There have been many proposals in the literature (e.g. Lifschitz [4], Erol, Nau and Subrahmanian [2], Bacchus and Yang [1]). These all have in common a reliance on metatheoretic operations on logical theories 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 this paper we provide a semantics for STRIPS-like systems in terms of a purely declarative situation calculus axiomatization for actions and their effects. On our view, STRIPS is a *mechanism* for computing the *progression* (Lin and Reiter [6], Pednault [8]) of an initial situation calculus database under the effects of an action. We illustrate this idea by describing two different STRIPS mechanisms, and proving their correctness with respect to their situation calculus specifications.

1 Preliminaries

The language $\mathcal{L}$ of the situation calculus is first-order, many-sorted, with sorts *state* for situations, *action* for actions, and *object* for everything else. It has the following domain independent predicates and functions: a constant S_0 of sort *state* denoting the initial state; a binary function $do(a, s)$ denoting the state resulting from performing the action a in the state s; a binary predicate $Poss(a, s)$ meaning that the action a is possible (executable) in state s; and a binary predicate $<$: *state* $\times$ *state*. $s < s'$ means that s' can be reached from s by a sequence of executable actions. We assume a finite number of *state independent* predicates with arity $object^n$, $n \geq 0$, a finite number of *state independent* functions 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 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 [7]), because we need induction on situations (Reiter [12]).

Often, we must restrict the situation calculus to a particular situation. For instance, the initial database is 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 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 $Poss$ or $<$.

We 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*.

Our situation calculus theory of actions, $\mathcal{D}$, will have the form (cf. Reiter [13] and Lin and Reiter [7]) $\mathcal{D} = \Sigma \cup \mathcal{D}_{ss} \cup \mathcal{D}_{ap} \cup \mathcal{D}_{una} \cup \mathcal{D}_{S_0}$, where

- Σ is the set of foundational axioms for the situation calculus (Lin and Reiter [6]). These play no role in the current paper, so we omit them.
- $\mathcal{D}_{ss}$ is a set of successor state axioms, one for each fluent $F(\vec{x}, s)$, of the form:

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

  where $\Phi_F(\vec{x}, a, s)$ is in $\mathcal{L}_s$. Successor state axioms embody a solution to the frame problem, as described in Reiter [11].
- $\mathcal{D}_{ap}$ is a set of action precondition axioms, one for each action $A(\vec{x})$, of the form:

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

  where $\Psi_A(\vec{x}, s)$ 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 $A(\vec{x})$, we have[1]

$$A(\vec{x}) = A(\vec{y}) \supset \vec{x} = \vec{y}.$$

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

[*]Fellow of the Canadian Institute for Advanced Research

[1]In what follows, $\vec{x} = \vec{y}$ is an abbreviation for $x_1 = y_1 \wedge \cdots \wedge x_n = y_n$.

Example 1.1 The following blocks world will provide a running example for this paper:

Actions

- $move(x, y, z)$: Move the block x from block y onto block z, provided both x and z are clear and block x is on top of block y.
- $movefromtable(x, y)$: Move the block x from the table onto block y, provided x is clear and on the table, and block y is clear.
- $movetotable(x, y)$: Move block x from block y onto the table, provided x is clear and x is on y.

Fluents

- $clear(x, s)$: Block x has no other blocks on top of it, in state s.
- $on(x, y, s)$: Block x is on (touching) block y, in state s.
- $ontable(x, s)$: Block x is on the table, in state s.

This setting can be axiomatized as follows:

Action Precondition Axioms

$$Poss(move(x, y, z), s) \equiv clear(x, s) \wedge clear(z, s) \wedge on(x, y, s) \wedge x \neq y \wedge x \neq z \wedge y \neq z,$$

$$Poss(movefromtable(x, y), s) \equiv clear(x, s) \wedge clear(y, s) \wedge ontable(x, s) \wedge x \neq y,$$

$$Poss(movetotable(x, y), s) \equiv clear(x, s) \wedge on(x, y, s) \wedge x \neq y.$$

Successor State Axioms

$$\begin{aligned}&Poss(a, s) \supset [clear(x, do(a, s)) \equiv \\ &\quad (\exists y, z)a = move(y, x, z) \vee (\exists y, z)a = move(x, y, z) \vee \\ &\quad (\exists y)a = movetotable(x, y) \vee (\exists y)a = movetotable(y, x) \vee \\ &\quad (\exists y)a = movefromtable(x, y) \vee \\ &\quad clear(x, s) \wedge \neg(\exists y, z)a = move(y, z, x) \wedge \\ &\qquad \neg(\exists y)a = movefromtable(y, x)],\end{aligned}$$

$$\begin{aligned}&Poss(a, s) \supset [on(x, y, do(a, s)) \equiv \\ &\quad (\exists z)a = move(x, z, y) \vee a = movefromtable(x, y) \vee \\ &\quad on(x, y, s) \wedge a \neq movetotable(x, y) \wedge \\ &\qquad \neg(\exists z)a = move(x, y, z)],\end{aligned}$$

$$\begin{aligned}&Poss(a, s) \supset \\ &\quad [ontable(x, do(a, s)) \equiv (\exists y)a = movetotable(x, y) \vee \\ &\qquad ontable(x, s) \wedge \neg(\exists y)a = movefromtable(x, y)].\end{aligned}$$

The central concept of (Lin and Reiter [6]) is this: Imagine performing an action α in an initial database $\mathcal{D}_{S_0}$. Let $\mathcal{S}_\alpha$ denote the resulting situation $do(\alpha, S_0)$. Imagine a database, $\mathcal{D}_{S_\alpha}$, which can act as a new initial database in the sense that

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 with respect to all possible future evolutions of the theory following α. $\mathcal{D}_{S_\alpha}$ is said to be a progression of $\mathcal{D}_{S_0}$ with respect to action α. To define progression formally, Lin and Reiter first introduce an equivalence relation over structures. Let M and M' be structures for $\mathcal{L}^2$ 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 fluent F, and every variable assignment σ,

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

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.

Definition 1.1 *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 (with respect to $\mathcal{D}$) 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 the new database is defined only up to logical equivalence. The new database is allowed to contain second-order sentences because, as shown in (Lin and Reiter [6]), progression is not always first order definable, but $\mathcal{D}_{S_\alpha}$ can always be captured by a set of second order sentences. However, Lin and Reiter specify some important special cases, which we shall exploit in this paper, for which progression is first order definable.

Our intuition about the semantics of STRIPS operators is that they are *mechanisms for progressing situation calculus databases.* This is the central idea of this paper, which is devoted to making this intuition precise.

2 STRIPS

Following Lifschitz ([4]), define an *operator description* to be a triple (P, D, A), where P is a sentence of a first order language $\mathcal{L}_{STRIPS}$ and D (the *delete list*) and A (the *add list*) are sets of sentences of $\mathcal{L}_{STRIPS}$. A *world description* W is any set of sentences of $\mathcal{L}_{STRIPS}$. A *STRIPS system* consists of:

1. A world description W_0, called the *initial* world description,
2. A binary relation $\rhd \subseteq 2^{\mathcal{L}_{STRIPS}} \times \mathcal{L}_{STRIPS}$,[2]
3. A set Op of symbols called *operators*, and
4. A family of operator descriptions $\{(P_\alpha, D_\alpha, A_\alpha)\}_{\alpha \in Op}$.

With each operator α is associated a world description W_α, the *successor world description of* W_0, defined by $W_\alpha = (W_0 - D_\alpha) \cup A_\alpha$. A successor world description W_α is *admissible* iff $W_0 \rhd P_\alpha$.

Sometimes, but not always, $\rhd$ will be the standard entailment relation for the first order language $\mathcal{L}_{STRIPS}$. In this case, admissibility simply corresponds to the fact that the precondition P_α is entailed by the initial world description W_0, in which case, on the standard view of STRIPS, the operator α is applicable. However, our intuitions about STRIPS are not standard, and we prefer to leave open the interpretation of the "entailment" relation $\rhd$.

[2]In his treatment of STRIPS, Lifschitz does not provide for the relation $\rhd$.

Our semantics for STRIPS systems is indirect; we define certain classes of theories in the situation calculus and show how to associate suitable STRIPS systems with those theories. Only STRIPS systems associated with such situation calculus theories will, on our account of STRIPS, be assigned a semantics. This leaves many STRIPS systems (namely those without an associated situation calculus theory) without a semantics; we are not very distressed by this, given that STRIPS systems, in their full generality, do not currently have coherent semantics anyway.

3 Two Versions of STRIPS

The STRIPS systems we derive apply only to a restricted class of situation calculus action theories for which the successor state axioms have a particular syntactic form, which we now define. A successor state axiom is *context free* iff it has the form:

$$\begin{aligned}&Poss(a,s) \supset [F(\vec{x}, do(a,s)) \equiv \\ &\quad (\exists \vec{v}^{(1)})a = A_1(\vec{\xi}^{(1)}) \vee \cdots \vee (\exists \vec{v}^{(m)})a = A_m(\vec{\xi}^{(m)}) \vee \\ &\quad F(\vec{x}, s) \wedge \neg(\exists \vec{w}^{(1)})a = B_1(\vec{\eta}^{(1)}) \wedge \cdots \wedge \\ &\qquad\qquad \neg(\exists \vec{w}^{(n)})a = B_n(\vec{\eta}^{(n)})].\end{aligned} \tag{1}$$

Here the A's and B's are function symbols of sort *action*, *not necessarily distinct from one another.* The $\vec{\xi}$ and $\vec{\eta}$ are sequences of distinct variables which *include all of the variables of* $\vec{x}$; the remaining variables of the $\vec{\xi}$ and $\vec{\eta}$ are those being existentially quantified by the $\vec{v}$ and $\vec{w}$, respectively. $\vec{x}$ could be the empty sequence.[3] The successor state axioms of our running blocks world example are context free. The following successor state axiom is not context free:

$$\begin{aligned}&Poss(a,s) \supset [ontable(x, do(a,s)) \equiv a = putontable(x) \vee \\ &\quad ontable(x) \wedge a \neq tiptable \wedge a \neq pickup(x)].\end{aligned}$$

This is because the action *tiptable* does not have x as a parameter.

The STRIPS systems which we shall characterize will be for languages $\mathcal{L}^2$ whose only function symbols of sort *object* are constants. Therefore, consider a ground action term α, and the context free successor state axiom (1) for fluent F, relativized to the initial state S_0. How does α affect the truth value of fluent F in the successor state $do(\alpha, S_0)$? By the unique names axioms for actions, together with the assumption that the successor state axioms are context free, this relativized axiom will be logically equivalent to a sentence of the form:

$$\begin{aligned}&Poss(\alpha, S_0) \supset \\ &\quad [F(\vec{x}, do(\alpha, S_0)) \equiv \vec{x} = \vec{X}^{(1)} \vee \cdots \vee \vec{x} = \vec{X}^{(m)} \vee \\ &\qquad F(\vec{x}, S_0) \wedge \vec{x} \neq \vec{Y}^{(1)} \wedge \cdots \wedge \vec{x} \neq \vec{Y}^{(n)}].\end{aligned}$$

Here the $\vec{X}$ and $\vec{Y}$ are tuples of constants of $\mathcal{L}^2$ obtained from those mentioned by the ground action term α. If we assume further that the action α is possible in the initial state, i.e. that $\mathcal{D} \models Poss(\alpha, S_0)$, this is equivalent to:

$$\begin{aligned}&F(\vec{x}, do(\alpha, S_0)) \equiv \vec{x} = \vec{X}^{(1)} \vee \cdots \vee \vec{x} = \vec{X}^{(m)} \vee \\ &\quad F(\vec{x}, S_0) \wedge \vec{x} \neq \vec{Y}^{(1)} \wedge \cdots \wedge \vec{x} \neq \vec{Y}^{(n)}.\end{aligned} \tag{2}$$

Example 3.1 Consider Example 1.1 under the "generic" ground action $move(X, Y, Z)$. The corresponding instances of (2) for the fluents *clear*, *on* and *ontable* are logically equivalent to:

$$\begin{aligned}&clear(x, do(move(X,Y,Z), S_0)) \equiv \\ &\qquad x = Y \vee x = X \vee clear(x, S_0) \wedge x \neq Z, \\ &on(x,y,do(move(X,Y,Z),S_0)) \equiv \\ &\qquad x = X \wedge y = Z \vee on(x,y,S_0) \wedge \neg[x = X \wedge y = Y], \\ &ontable(x, do(move(X,Y,Z),S_0)) \equiv ontable(x, S_0).\end{aligned}$$

For the generic ground actions $movefromtable(X, Y)$ and $movetotable(X, Y)$ we obtain:

$$\begin{aligned}&clear(x, do(movefromtable(X,Y), S_0)) \equiv \\ &\qquad x = X \vee clear(x, S_0) \wedge x \neq Y, \\ &on(x,y,do(movefromtable(X,Y),S_0)) \equiv \\ &\qquad x = X \wedge y = Y \vee on(x,y,S_0), \\ &ontable(x, do(movefromtable(X,Y),S_0)) \equiv \\ &\qquad ontable(x, S_0) \wedge x \neq X, \\ &clear(x, do(movetotable(X,Y),S_0)) \equiv \\ &\qquad x = X \vee x = Y \vee clear(x, S_0), \\ &on(x,y,do(movetotable(X,Y),S_0)) \equiv \\ &\qquad on(x,y,S_0) \wedge \neg[x = X \wedge y = Y], \\ &ontable(x, do(movetotable(X,Y),S_0)) \equiv \\ &\qquad x = X \vee ontable(x, S_0).\end{aligned}$$

3.1 OCF-STRIPS: Open World, Context Free STRIPS

Our point of departure is an action theory $\mathcal{D} = \Sigma \cup \mathcal{D}_{ss} \cup \mathcal{D}_{ap} \cup \mathcal{D}_{una} \cup \mathcal{D}_{S_0}$, with the following properties:

1. The only function symbols of sort *object* that the second order language $\mathcal{L}^2$ possesses are constants.[4]
2. Each state dependent sentence of $\mathcal{D}_{S_0}$ is a ground fluent literal, i.e. of the form $F(\vec{C}, S_0)$ or $\neg F(\vec{C}, S_0)$ for fluent F and constants $\vec{C}$ of sort *object*.
3. $\mathcal{D}_{S_0}$ contains *unique names axioms* for constants of sort *object*: For each pair of distinct constant names C and C' of sort *object*, the axiom $C \neq C'$.
4. $\mathcal{D}_{S_0}$ contains no pair of complementary literals.
5. Each successor state axiom of $\mathcal{D}_{ss}$ is context free.
6. We are progressing with respect to α, a ground action term, and α is possible initially:
$$\mathcal{D} \models Poss(\alpha, S_0).$$
7. For each fluent F, the following consistency condition (Reiter [11]) is satisfied:
$$\begin{aligned}\mathcal{D}_{ap} \cup \mathcal{D}_{una} \models \neg(\exists \vec{x}, a, s).Poss(a,s) \wedge \\ \gamma_F^+(\vec{x}, a, s) \wedge \gamma_F^-(\vec{x}, a, s),\end{aligned} \tag{3}$$
where F's successor state axiom has the form
$$\begin{aligned}&Poss(a,s) \supset [F(\vec{x}, do(a,s)) \equiv \\ &\qquad \gamma_F^+(\vec{x}, a, s) \vee F(\vec{x}, s) \wedge \neg\gamma_F^-(\vec{x}, a, s)].\end{aligned} \tag{4}$$

[3]Notice that this is a slightly more restricted definition for context free successor state axioms than that of (Lin and Reiter [6]). Specifically, we did not require that each action term $A_i(\vec{\xi}^{(i)})$ and $B_j(\vec{\eta}^{(j)})$ mention all of the variables of $\vec{x}$ in the variables $\vec{\xi}^{(i)}$ and $\vec{\eta}^{(j)}$.

[4]Recall that $\mathcal{L}^2$ is the language in which $\mathcal{D}$ is expressed.

The consistency condition (3) deserves a brief explanation. Following Pednault [9] and Schubert [14], Reiter [11] provides a solution to the frame problem in the absence of state constraints which syntactically transforms a pair of effect axioms for a given fluent F into a successor state axiom for F. The effect axioms are assumed to have the syntactic forms:

$$Poss(a,s) \wedge \gamma_F^+(\vec{x},a,s) \supset F(\vec{x},do(a,s)),$$

and

$$Poss(a,s) \wedge \gamma_F^-(\vec{x},a,s) \supset \neg F(\vec{x},do(a,s)).$$

Reiter applies the *explanation closure* idea of Schubert [14] to obtain the following frame axioms for F:

$$Poss(a,s) \wedge \neg F(\vec{x},s) \wedge F(\vec{x},do(a,s)) \supset \gamma_F^+(\vec{x},a,s),$$

$$Poss(a,s) \wedge F(\vec{x},s) \wedge \neg F(\vec{x},do(a,s)) \supset \gamma_F^-(\vec{x},a,s).$$

The successor state axiom (4) is logically equivalent to the conjunction of the above four sentences, whenever the consistency condition (3) holds. Notice that the consistency condition makes good sense: If it were violated, so that for some $\vec{X}, A, S$ we have $Poss(A,S)$, $\gamma_F^+(\vec{X},A,S)$, and $\gamma_F^-(\vec{X},A,S)$, then we could derive an immediate inconsistency from the above two effect axioms.

It is easy (but tedious) to verify that each fluent of Example 1.1 satisfies the consistency condition.

In keeping with our intuition that STRIPS systems are mechanisms for progressing situation calculus databases, we want now to characterize the result of progressing $\mathcal{D}_{S_0}$ under the effects of the ground action α in the case of action theories of the above kind. This turns out to be easy, since the necessary work has already been done (Lin and Reiter [6], Section 6).

Let $\mathcal{S}$ be the following set of sentences:

1. Initialize $\mathcal{S}$ to $\{\varphi \in \mathcal{D}_{S_0} \mid \varphi \text{ is state independent}\}$.
2. For each fluent F do (with reference to the instance (2) of F's successor state axiom):
   (a) Add to $\mathcal{S}$ the sentence $F(\vec{X}^{(i)}, do(\alpha, S_0))$, $i = 1, \ldots, m$.
   (b) For each ground instance $F(\vec{C}, S_0) \in \mathcal{D}_{S_0}$ add to $\mathcal{S}$ the sentence $F(\vec{C}, do(\alpha, S_0))$, whenever $\vec{C}$ is a tuple of constants different from each $\vec{Y}^{(i)}$, $i = 1, \ldots, n$. (Here, we invoke the unique names axioms for constants of sort *object*).
   (c) Add to $\mathcal{S}$ the sentence $\neg F(\vec{Y}^{(i)}, do(\alpha, S_0))$, $i = 1, \ldots, n$.
   (d) For each ground instance $\neg F(\vec{C}, S_0) \in \mathcal{D}_{S_0}$ add to $\mathcal{S}$ the sentence $\neg F(\vec{C}, do(\alpha, S_0))$, whenever $\vec{C}$ is a tuple of constants different from each $\vec{X}^{(i)}$, $i = 1, \ldots, m$. (We again invoke the unique names axioms for constants of sort *object*).

The resulting set $\mathcal{S}$ enjoys the property that $\mathcal{S} \cup \mathcal{D}_{una}$ is a progression of $\mathcal{D}_{S_0}$ under action α (Lin and Reiter [6], Theorem 4).[5] Moreover, the state dependent sentences of $\mathcal{S}$ are all ground literals, and the results of Lin and Reiter [6] guarantee that $\mathcal{S}$ contains no pair of complementary literals. It follows that $\mathcal{S}$ can serve as a new initial database for the purposes of iterating the above progression mechanism.

Now we interpret the above construction of the set $\mathcal{S}$ as a STRIPS operator. Imagine suppressing the state argument S_0 of all the ground literals of $\mathcal{D}_{S_0}$. Now ask what sequence of deletions and additions of ground literals must be performed on the state-suppressed version of $\mathcal{D}_{S_0}$ in order to obtain the state-suppressed version of $\mathcal{S}$ (i.e. $\mathcal{S}$ with the state argument $do(\alpha, S_0)$ suppressed in its sentences). The deletions and additions necessary to achieve this state-suppressed transformation of $\mathcal{D}_{S_0}$ to $\mathcal{S}$ will define the delete and add lists for the STRIPS operator α.

It is easy to see that the following deletions and additions, when applied to $\mathcal{D}_0$, the state-suppressed version of $\mathcal{D}_{S_0}$, yields the state-suppressed version of $\mathcal{S}$:

For each fluent F do (with reference to the instance (2) of F's successor state axiom):

1. Delete from $\mathcal{D}_0$ the sentences $\neg F(\vec{X}^{(i)})$, $i = 1, \ldots, m$.
2. Delete from $\mathcal{D}_0$ the sentences $F(\vec{Y}^{(i)})$, $i = 1, \ldots, n$.
3. Add to $\mathcal{D}_0$ the sentences $F(\vec{X}^{(i)})$, $i = 1, \ldots, m$.
4. Add to $\mathcal{D}_0$ the sentences $\neg F(\vec{Y}^{(i)})$, $i = 1, \ldots, n$.

It is now clear how to define a STRIPS system and its associated operator for α:[6]

1. The language $\mathcal{L}_{STRIPS}$ is the state-suppressed version of $\mathcal{L}^2$.[7]
2. The initial world description is $\mathcal{D}_0$.
3. $\rhd$ is ordinary logical entailment; for a world description W and sentence $\sigma \in \mathcal{L}_{STRIPS}$, $W \rhd \sigma$ iff $W \models \sigma$.
4. α's precondition is the state-suppressed version of the right hand side of the equivalence in α's situation calculus action precondition axiom.
5. For each fluent F, include in α's add and delete lists those literals specified above for obtaining the state suppressed version of $\mathcal{S}$.

To our knowledge, OCF-STRIPS is the only variant of STRIPS which specifically provides for an incomplete database of ground literals, and which is provably correct with respect to a logical specification.

Example 3.2 Continuing with our blocks world example, we can "read off" the OCF-STRIPS operator schema for *move* from the instances of the successor state axioms given in Example 3.1:

[5]The consistency condition (3) was inadvertently omitted from the assumptions underlying Theorem 4 of Lin and Reiter [6].

[6]See Section 2 for the relevant definitions.

[7]We take it as self evident what is meant formally by the language obtained by suppressing objects of sort *state* from the language $\mathcal{L}^2$.

$move(X,Y,Z)$[8]

P: $clear(X) \wedge clear(Z) \wedge on(X,Y) \wedge$
$X \neq Z \wedge X \neq Y \wedge Y \neq Z.$
D: $\neg clear(Y), \neg clear(X), clear(Z),$
$\neg on(X,Z), on(X,Y).$
A: $clear(Y), clear(X), \neg clear(Z),$
$on(X,Z), \neg on(X,Y).$

The operator description schemas for $movefromtable$ and $movetotable$ are obtained in the same way:

$movefromtable(X,Y)$

P: $clear(X) \wedge clear(Y) \wedge ontable(X) \wedge X \neq Y.$
D: $\neg clear(X), \neg on(X,Y), ontable(X), clear(Y).$
A: $clear(X), on(X,Y), \neg ontable(X), \neg clear(Y).$

$movetotable(X,Y)$

P: $clear(X) \wedge on(X,Y) \wedge X \neq Y.$
D: $\neg clear(X), \neg clear(Y), on(X,Y), \neg ontable(X).$
A: $clear(X), clear(Y), \neg on(X,Y), ontable(X).$

3.2 RCF-STRIPS: Relational, Context Free STRIPS

This version of STRIPS derives from action theories $\mathcal{D}$ of the form $\mathcal{D} = \Sigma \cup \mathcal{D}_{ss} \cup \mathcal{D}_{ap} \cup \mathcal{D}_{una} \cup \mathcal{D}_{S_0}$, with the following properties:

1. The only function symbols of sort *object* that the second order language $\mathcal{L}^2$ possesses are constants.
2. $\mathcal{D}_{S_0}$ contains one sentence of the following form, for each fluent F:
$$F(\vec{x}, S_0) \equiv \vec{x} = \vec{C}^{(1)} \vee \cdots \vee \vec{x} = \vec{C}^{(n)}, \qquad (5)$$
where the $\vec{C}^{(i)}$ are tuples of constant symbols of sort *object*. These are the only state dependent sentences of $\mathcal{D}_{S_0}$. (Initial databases of this form are special cases of the *relatively complete* databases defined in (Lin and Reiter [6]).) The case $n = 0$ is permitted, in which case this axiom is $F(\vec{x}, S_0) \equiv false$. For example, if an agent's hand is initially empty:
$$holding(x, S_0) \equiv false.$$
If initially, block A is on B, D is on A, C is on E, and no other block is on a block:
$$on(x,y,S_0) \equiv$$
$$x = A \wedge y = B \vee x = D \wedge y = A \vee x = C \wedge y = E.$$
3. $\mathcal{D}_{S_0}$ contains *unique names axioms* for constants of sort *object*.
4. Each successor state axiom of $\mathcal{D}_{ss}$ is *context free*.
5. We are progressing with respect to α, a ground action term, and α is possible initially:
$$\mathcal{D} \models Poss(\alpha, S_0).$$

Notice that the single sentence (5) is logically equivalent to:
$$F(\vec{C}^{(1)}, S_0), \ldots, F(\vec{C}^{(n)}, S_0), \qquad (6)$$
$$\vec{x} \neq \vec{C}^{(1)} \wedge \cdots \wedge \vec{x} \neq \vec{C}^{(n)} \supset \neg F(\vec{x}, S_0). \qquad (7)$$
Notice also that, given all the positive instances (6) of F, we can trivially determine the sentence (7). So it is sufficient to *represent* a database of this form (say for computational purposes) by the set of all positive instances of F. This, we claim, is what some versions of STRIPS do (but suppressing the state argument). This is also what relational databases do; in fact, the unique names assumption together with the condition (5) on $\mathcal{D}_{S_0}$ are the defining properties for a *relational database* (Reiter [10]). The relational *tables* are just the ground instances of the fluents F. (But bear in mind that *logically*, the database consists of the table for F, together with the axiom (7) and unique names axioms.)

As we did in the previous section, we want now to characterize the result of progressing $\mathcal{D}_{S_0}$ under the effects of the ground action α in the case of action theories of the above kind. To do so, we appeal to the results in (Lin and Reiter [6], Section 5). Consider the context free successor state axiom (2) for fluent F which we relativized to the initial state S_0. By our assumption (5) on the syntactic form of $\mathcal{D}_{S_0}$, (2) is equivalent to:
$$F(\vec{x}, do(\alpha, S_0)) \equiv \vec{x} = \vec{X}^{(1)} \vee \cdots \vee \vec{x} = \vec{X}^{(m)} \vee$$
$$[\vec{x} = \vec{C}^{(1)} \vee \cdots \vee \vec{x} = \vec{C}^{(n)}] \wedge \vec{x} \neq \vec{Y}^{(1)} \wedge \cdots \wedge \vec{x} \neq \vec{Y}^{(n)}.$$
Let $\vec{C}^{(1)}, \ldots, \vec{C}^{(r)}$ be *all* the $\vec{C}^{(k)}$ that are different tuples than *all* of the $\vec{Y}^{(i)}$. Then, by unique names axioms for constant symbols of sort *object*, the above sentence will be logically equivalent to
$$\begin{array}{c} F(\vec{x}, do(\alpha, S_0)) \equiv \vec{x} = \vec{X}^{(1)} \vee \cdots \vee \vec{x} = \vec{X}^{(m)} \vee \\ \vec{x} = \vec{C}^{(1)} \vee \cdots \vee \vec{x} = \vec{C}^{(r)}. \end{array} \qquad (8)$$
Let $\mathcal{S}$ be the following set of sentences:

1. Initialize $\mathcal{S}$ to $\{\varphi \in \mathcal{D}_{S_0} \mid \varphi \text{ is state independent}\}$.
2. For each fluent F do: Add the sentence (8) to $\mathcal{S}$.

The resulting set $\mathcal{S}$ enjoys the property that $\mathcal{S} \cup \mathcal{D}_{una}$ is a progression of $\mathcal{D}_{S_0}$ under action α (Lin and Reiter [6], Theorem 3). Moreover, $\mathcal{S}$ has the same syntactic form as $\mathcal{D}_{S_0}$, and so can serve as a new initial database for the purposes of iterating the above progression mechanism.

Now we interpret the above construction of the set $\mathcal{S}$ as a STRIPS operator. Imagine representing the state dependent sentences
$$F(\vec{x}, S_0) \equiv \vec{x} = \vec{C}^{(1)} \vee \cdots \vee \vec{x} = \vec{C}^{(n)} \qquad (9)$$
by the state-suppressed relational database of ground instances $F(\vec{C}^{(1)}), \ldots, F(\vec{C}^{(n)})$. We emphasize that this representation is merely a shorthand for the sentence (9). Now ask what sequence of deletions and additions of ground literals must be performed on $\mathcal{D}_0$, the state-suppressed relational database version of $\mathcal{D}_{S_0}$ in order to obtain the state-suppressed relational version of $\mathcal{S}$. The deletions and additions necessary to achieve this transformation of $\mathcal{D}_0$ to the corresponding representation of $\mathcal{S}$ will define the delete and add lists for the STRIPS operator α.

It is easy to see that the following deletions and additions, when applied to $\mathcal{D}_0$, yield the state-suppressed, relational database representation of $\mathcal{S}$:

For each fluent F do (with reference to (2)):

[8]Notice that these are *schemas*, standing for the family of operators obtained by instantiating the "variables" X, Y and Z of the schema by constants of our situation calculus language.

1. Delete from $\mathcal{D}_0$ the sentences $F(\vec{Y}^{(i)})$, $i = 1, \ldots, n$.
2. Add to $\mathcal{D}_0$ the sentences $F(\vec{X}^{(i)})$, $i = 1, \ldots, m$.

It is now clear how to define a STRIPS system and its associated operator for α:[9]

1. The language $\mathcal{L}_{STRIPS}$ is the state-suppressed version of $\mathcal{L}^2$.
2. The initial world description is $\mathcal{D}_0$.
3. For a sentence $\sigma \in \mathcal{L}_{STRIPS}$, $W \rhd \sigma$ iff $\mathcal{R}(W) \models \sigma$. Here, W is a world description in relational database form for all its fluents, i.e. the only sentences in W that mention a fluent are ground atoms of that fluent. $\mathcal{R}(W)$ is the translation of the relational database part of W to its full logical form as follows: $\mathcal{R}(W)$ consists of the sentences of W that do not mention a fluent, together with those sentences of the form
$$F(\vec{x}) \equiv \vec{x} = \vec{C}^{(1)} \vee \cdots \vee \vec{x} = \vec{C}^{(n)}$$
where $F(\vec{C}^{(1)}), \ldots, F(\vec{C}^{(n)})$ are *all* the ground instances of a fluent F in W.
4. α's precondition is the state-suppressed version of the right hand side of the equivalence in α's situation calculus action precondition axiom.
5. For each fluent F, include in α's add and delete lists those literals specified above for obtaining the state suppressed relational database representation of $\mathcal{S}$.

Example 3.3 Consider the same actions, fluents and axioms as in Example 1.1, except treat this setting now as an instance of an RCF-STRIPS situation calculus specification. In this case, as before, we can "read off" the RCF-STRIPS operator schema for *move* from the instances of the successor state axioms of Example 3.1:

$move(X, Y, Z)$

P: $clear(X) \wedge clear(Z) \wedge on(X, Y) \wedge X \neq Z \wedge X \neq Y \wedge Y \neq Z$.
D: $clear(Z), on(X, Y)$.
A: $clear(Y), clear(X), on(X, Z)$.

The operator description schemas for *movefromtable* and *movetotable* are obtained in the same way:

$movefromtable(X, Y)$

P: $clear(X) \wedge clear(Y) \wedge ontable(X) \wedge X \neq Y$.
D: $clear(Y), ontable(X)$.
A: $clear(X), on(X, Y)$.

$movetotable(X, Y)$

P: $clear(X) \wedge on(X, Y) \wedge X \neq Y$.
D: $on(X, Y)$.
A: $clear(X), clear(Y), ontable(X)$.

4 Pednault's ADL

The only prior literature similar to our progression semantics for STRIPS-like systems is by Pednault ([9], [8]). Like us, Pednault relates a STRIPS database to the initial state of a situation calculus axiomatization. But our interpretation of such a database, namely as a state-suppressed situation calculus *theory*, distinguishes our approach from Pednault's, in which these databases are first order *structures*. So for Pednault, STRIPS is a mapping from first order structures to first order structures, where this mapping is defined by the addition and deletion of tuples applied to the relations of the structure. ADL, Pednault's generalization of STRIPS, is just such a mapping between structures. For us, as for Lifschitz [4], STRIPS is a mapping from first order theories to (possibly second order) theories, where this mapping is effected by add and delete lists of *sentences* applied to the theory. The problem with the ADL view on STRIPS is that it does not provide a feasible mechanism for applying a STRIPS operator in the case that the database is a logically incomplete theory (e.g. OCF-STRIPS of Section 3.1). For in such a case, every model of this theory must be mapped by an ADL operator into its transformed structure, and it is the set of all such transformed structures which represents the effect of the ADL operator. When there are infinitely many such models, or even when they are finite in number but numerous, ADL becomes an unattractive STRIPS mechanism. In contrast, our focus is on STRIPS mechanisms that operate on logical theories, and hence operate on the single sentential representations of these many models.

5 Conclusions

1. On our view STRIPS is a *mechanism* for progressing a situation calculus theory, and its semantics can best be understood with reference to a suitable situation calculus axiomatization of actions and their effects.
2. With this notion of progression in hand, it becomes possible to formulate various STRIPS-like systems, and prove their correctness with respect to our progression semantics. In this paper we have done just that for two different STRIPS systems (OCF and RCF-STRIPS). In this connection OCF-STRIPS is of particular interest because it provides for a (limited) form of logical incompleteness of the database.
3. Notice that it is a completely mechanical process to obtain the OCF-STRIPS operators from a situation calculus axiomatization of some domain. Similarly for RCF-STRIPS. In other words, these purely declarative situation calculus specifications can be *compiled* into appropriate STRIPS systems.
4. The connection of RCF-STRIPS to relational databases (Section 3.2) suggests a natural generalization of STRIPS operators to allow for arbitrary relational algebra operators (not just adds and deletes) in defining the operator's effects. This can indeed be done, and an appropriate semantics defined in terms of a situation calculus axiomatization that relaxes the context free restriction on successor state axioms of Section 3.2 (Lin and Reiter [5]: the full version of this paper).
5. We have considered only STRIPS systems that compute the full result of progression. Sometimes, for

[9] See Section 2 for the relevant definitions.

instance for computational purposes, it may be better to compute only that part of the progression that is relevant to the goals of interest. For example, if our blocks world includes a fluent for the colors of blocks, then there are no need to progress this fluent if our goals have nothing to do with colors.

Formally, we say that a given STRIPS system is *sound* with respect to a consistent action theory $\mathcal{D} = \Sigma \cup \mathcal{D}_{ss} \cup \mathcal{D}_{ap} \cup \mathcal{D}_{una} \cup \mathcal{D}_{S_0}$ iff: For any finite (including empty) sequence δ of operators in the STRIPS system, there is a corresponding finite sequence $\mathbf{T}$ of action in the situation calculus such that if the world description W_δ after performing δ on W_0 is admissible, then $\mathbf{T}$ is executable in S_0 and $\mathcal{D}_{\mathbf{T}} \models Th(W_\delta)$, where $\mathcal{D}_{\mathbf{T}}$ is the result of suppressing states in the progression of $\mathcal{D}_{S_0}$ to $S_{\mathbf{T}}$, and $Th(W_\delta) = \{\varphi \mid W_\delta \triangleright \varphi\}$.

Generally, if a STRIPS system is sound with respect to an action theory, then any goal that is achievable in the STRIPS system is also achievable in the action theory. However, the converse is not true in general. We say a STRIPS system is *adequate* for a given goal if the converse is true, i.e. whenever the goal is achievable in the situation calculus theory by a sequence of actions, it is also achievable by a sequence of operators in the STRIPS system.

With these definitions in hand, we can show (Lin and Reiter [5]: the full version of this paper) that, with respect to a suitable action theory, the main example STRIPS system considered in (Fikes and Nilsson [3]) is both sound, and adequate for the class of goals considered there. This is of interest because it shows that our semantics is sophisticated enough to handle the concepts of non-literal and/or non-essential formulas considered in Lifschitz's [4] analysis of this same example.

Acknowledgements:

Thanks to the other members of the University of Toronto Cognitive Robotics Group (Yves Lespérance, Hector Levesque, Richard Mann, Daniel Marcu and Richard Scherl) for their comments and suggestions. This research was supported by grants from the Natural Sciences and Engineering Research Council of Canada, the Institute for Robotics and Intelligent Systems of the Government of Canada, and the Information Technology Research Centre of the Government of Ontario.

References

[1] F. Bacchus and Q. Yang. Downward refinement and the efficiency of hierarchical problem solving. *Artificial Intelligence*, 1995. To appear.

[2] K. Erol, D.S. Nau, and V.S. Subrahmanian. On the complexity of domain-independent planning. In *Proceedings of the National Conference on Artificial Intelligence*, pages 381–386, San Jose, CA, 1992. American Association for Artificial Intelligence.

[3] R.E. Fikes and N.J. Nilsson. STRIPS: a new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3/4):189–208, 1971.

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

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

[6] F. Lin and R. Reiter. How to progress a database (and why) I. Logical foundations. In Jon Doyle, Erik Sandewall, and Pietro Torasso, editors, *Proc. KR'94, Fourth Int. Conf. on Principles of Knowledge Representation and Reasoning*, pages 425–436, 1994.

[7] F. Lin and R. Reiter. State constraints revisited. *J. of Logic and Computation, special issue on actions and processes*, 4:655–678, 1994.

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

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

[10] R. Reiter. Towards a logical reconstruction of relational database theory. In M.L. Brodie, J. Mylopoulos, and J.W. Schmidt, editors, *On Conceptual Modelling: Perspectives from Artificial Intelligence, Databases and Programming Languages*, pages 191–233. Springer, New York, 1984.

[11] 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.

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

[13] R. Reiter. On specifying database updates. *Journal of Logic Programming*, to appear.

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