Some Contributions to the Metatheory of the Situation Calculus

Fiora Pirri
Dipartimento di Informatica e Sistemistica
Università degli Studi di Roma "La Sapienza"
Via Salaria 113, 00198 Roma, Italy
pirri@assi.dis.uniroma1.it

Ray Reiter
Department of Computer Science
University of Toronto
Toronto, Canada M5S 1A4
reiter@ai.toronto.edu

December 28, 1998

Abstract

We focus on a rich axiomatization for actions in the situation calculus that includes, among other features, a solution to the frame problem for deterministic actions. Our work is foundational in nature, directed at simplifying the entailment problem for these axioms. Specifically, we make four contributions to the metatheory of situation calculus axiomatizations of dynamical systems:

1. We prove that the above-mentioned axiomatization for actions has a relative satisfiability property; the full axiomatization is satisfiable iff the axioms for the initial state are.
2. We define the concept of regression relative to these axioms, and prove a soundness and completeness theorem for a regression-based approach to the entailment problem for a wide class of queries.
3. Our formalization of the situation calculus requires certain foundational axioms specifying the domain of situations. These include an induction axiom, whose presence complicates human and automated reasoning in the situation calculus. We characterize various classes of sentences whose proofs do not require induction, and in some cases, some of the other foundational axioms.
4. We prove that the logic programming language GOLOG never requires any of the foundational axioms for the evaluation of programs.

1 Introduction

The situation calculus ([22]) has been a part of the artificial intelligence zeitgeist almost from the very beginning of the field. It is included in the standard material of every introductory course on AI, and it is the language of choice for investigations of various technical problems that arise in theorizing about actions and their effects (e.g. [19], [8], [13]). But only recently has it been taken seriously as a foundation for practical work in planning, control, simulation, database updates, agent programming and robotics (e.g. [18, 9, 17, 34, 33, 2, 14, 16]). In parallel with these developments of its applications, there have emerged axiomatizations for the situation calculus, and explorations of some of their metamathematical and computational properties (e.g. [32, 21]). This paper continues these explorations, focusing on a rich axiomatization for actions in the situation calculus that includes, among other features, a solution to the frame problem for deterministic actions. Our work is foundational in nature, directed at simplifying the entailment problem for these axioms. Specifically, we make four contributions to the metatheory of situation calculus axiomatizations of dynamical systems:

1. We prove that the above-mentioned axiomatization for actions has a relative satisfiability property; the full axiomatization is satisfiable iff the axioms for the initial state are.

2. We define the concept of regression relative to these axioms, and prove a soundness and completeness theorem for a regression-based approach to the entailment problem for a wide class of queries.

3. Our formalization of the situation calculus requires certain foundational axioms specifying the domain of situations. These include an induction axiom, whose presence complicates human and automated reasoning in the situation calculus. We characterize various classes of sentences whose proofs do not require induction, and in some cases, some of the other foundational axioms.

4. We prove that the logic programming language GOLOG [18] never requires any of the foundational axioms for the evaluation of programs.

2 Formal Preliminaries

2.1 The Language of the Situation Calculus

The language $\mathcal{L}_{sitcalc}$ that we adopt in this paper is a second order language with equality. It has three disjoint sorts: *action* for actions, *situation* for situations, and a catch-all sort *object* for everything else depending on the domain of application. Apart from the standard alphabet of logical symbols – we use $\wedge$, $\neg$ and $\exists$, with the usual definitions of a full set of connectives and quantifiers – $\mathcal{L}_{sitcalc}$ has the following alphabet:

- Countably infinitely many individual variable symbols of each sort. We shall use s and a, with subscripts and superscripts, for variables of sort *situation* and *action*, respectively. We normally use lower case roman letters other than a, s, with subscripts and superscripts for variables of sort *object*. In addition, because $\mathcal{L}_{sitcalc}$ is second order, its alphabet includes countably infinitely many predicate variables of all arities.

- Two function symbols of sort *situation*:

    1. A constant symbol S_0, denoting the initial situation.
    2. A binary function symbol $do : action \times situation \to situation$. The intended interpretation is that $do(a, s)$ denotes the successor situation resulting from performing action a in situation s.

- A binary predicate symbol $\sqsubset$: $situation \times situation$, defining an ordering relation on situations. The intended interpretation of situations is as action histories, in which case $s \sqsubset s'$ means that s is a proper subhistory of s'.

- A binary predicate symbol $Poss : action \times situation$. The intended interpretation of $Poss(a, s)$ is that it is possible to perform the action a in situation s.

- For each $n \geq 0$, countably infinitely many predicate symbols with arity n, and sorts $(action \cup object)^n$. These are used to denote situation independent relations like $human(John)$, $primeNumber(n)$, $movingAction(run(person, loc1, loc2))$, etc.

- For each $n \geq 0$, countably infinitely many function symbols of sort $(action \cup object)^n \to object$. These are used to denote situation independent functions like $sqrt(x)$, $height(MtEverest)$, $agent(run(person, loc1, loc2))$, etc.

    For each $n \geq 0$, a finite or countably infinite number of function symbols of sort $(action \cup object)^n \to action$. These are called *action functions*, and are used to denote actions like $pickup(x)$, $move(A, B)$, etc. In most applications, there will be just finitely many action functions, but we allow the possibility of an infinite number of them.

    Notice that we distinguish between function symbols taking values of sort *object* and those – the action functions – taking values of sort *action*. In what follows, the latter will be distinguished by the requirement that they be axiomatized in a particular way by what we shall call *action precondition axioms*.

- For each $n \geq 0$, a finite or countably infinite number of predicate symbols with arity $n+1$, and sorts $(action \cup object)^n \times situation$. These predicate symbols are called *relational fluents*. In most applications, there will be just finitely many relational fluents, but we do not preclude the possibility of an infinite number of them. These are used to denote situation dependent relations like $ontable(x, s)$, $husband(Mary, John, s)$, etc. Notice that relational fluents take just one argument of sort *situation*, and this is always its last argument.

- For each $n \geq 0$, a finite or countably infinite number of function symbols of sort $(action \cup object)^n \times situation \rightarrow action \cup object$. These function symbols are called *functional fluents*. In most applications, there will be just finitely many functional fluents, but we do not preclude the possibility of an infinite number of them. These are used to denote situation dependent functions like $age(Mary, s)$, $primeMinister(Italy, s)$, etc. Notice that functional fluents take just one argument of sort *situation*, and this is always its last argument.

Notice that only two function symbols of $\mathcal{L}_{sitcalc}$ – S_0 and do – are permitted to take values in sort *situation*.

2.2 Foundational Axioms for the Situation Calculus

Here, we focus on the domain of situations. The primary intuition about situations that we wish to capture axiomatically is that they are finite sequences of actions. We want also to be able to say that a certain sequence of actions is a subsequence of another. By modifying earlier proposals of Reiter [33], Lin and Reiter [20] and Pinto [29], we adopt the following four foundational axioms for the situation calculus.[1]

$$do(a_1, s_1) = do(a_2, s_2) \supset a_1 = a_2 \wedge s_1 = s_2, \tag{1}$$
$$(\forall P).P(S_0) \wedge (\forall a, s)[P(s) \supset P(do(a, s))] \supset (\forall s)P(s). \tag{2}$$

Axiom (1) is a unique names axiom for situations; two situations are the same iff they are the same sequence of actions. Two situations S_1 and S_2 may be different, yet assign the same truth values to all fluents. So a situation in the version of the situation calculus presented here must not be identified with the set of fluents that hold in that situation, i.e with a state. The proper way to understand a situation is as a *history*, namely, a sequence of actions; two situations are equal iff they denote identical histories. The second axiom (2) is second order induction on situations. The importance of induction for the situation calculus is described by Reiter [32].

There are two more axioms:

$$\neg s \sqsubset S_0, \tag{3}$$
$$s \sqsubset do(a, s') \equiv s \sqsubseteq s'. \tag{4}$$

Here $s \sqsubseteq s'$ is an abbreviation for $s \sqsubset s' \vee s = s'$. The relation $\sqsubset$ provides an ordering relation on situations. Intuitively, $s \sqsubset s'$ means that the action sequence s' can be obtained from the sequence s by adding one or more actions to the front of s.[2]

[1] In what follows, lower case Roman characters will denote variables in formulas. Moreover, free variables will always be implicitly universally prenex quantified.

[2] Readers familiar with the programming language LISP will have noticed that in the situation calculus, the constant S_0 is just like NIL, and do acts like $cons$. Situations are simply *lists* of primitive actions. For example, the situation term $do(C, do(B, do(A, S_0)))$ is simply an alternative syntax for the LISP list $(C\ B\ A)$ $(= cons(C, cons(B, cons(A, nil))))$. Notice that to obtain the action *history* corresponding to this

The above four axioms are *domain independent*. They provide the basic properties of situations in any domain specific axiomatization of particular fluents and actions. Henceforth, we shall call them Σ.[3]

2.3 Basic Theories of Actions

Our concern in this paper will be with axiomatizations for actions and their effects that have a particular syntactic form. These are called *basic action theories*, and we next describe these.

Definition 2.1 The Uniform Formulas
Let σ be a term of sort *situation*. The terms of $\mathcal{L}_{sitcalc}$ *uniform in* σ are the smallest set of terms such that:

1. Any term that does not mention a term of sort *situation* is uniform in σ.

2. σ is uniform in σ.

3. If g is an n-ary function symbol other than do and S_0, and $t_1, \ldots, t_n$ are terms uniform in σ whose sorts are appropriate for g, then $g(t_1, \ldots, t_n)$ is a term uniform in σ.

The formulas of $\mathcal{L}_{sitcalc}$ *uniform in* σ are the smallest set of formulas such that:

1. If t_1 and t_2 are terms of the same sort *object* or *action*, and if they are both uniform in σ, then $t_1 = t_2$ is a formula uniform in σ.

2. When P is an n-ary predicate symbol of $\mathcal{L}_{sitcalc}$, other than $Poss$ and $\sqsubset$, and $t_1, \ldots, t_n$ are terms uniform in σ whose sorts are appropriate for P, then $P(t_1, \ldots, t_n)$ is a formula uniform in σ.

3. Whenever U_1, U_2 are formulas uniform in σ, so are $\neg U_1$, $U_1 \wedge U_2$ and $(\exists v)U_1$ provided v is an individual variable, and it is not of sort *situation*.

Thus, a formula of $\mathcal{L}_{sitcalc}$ is uniform in σ iff it is first order, it does not mention the predicates $Poss$ or $\sqsubset$, it does not quantify over variables of sort *situation*, it does not mention equality on situations, and whenever it mentions a term of sort *situation* in the situation argument position of a fluent, then that term is σ.

term, namely the performance of action A, followed by B, followed by C, we read this list from right to left. Therefore, when one reads situation terms from right to left, the relation $s \sqsubset s'$ means that situation s is a proper subhistory of the situation s'. The situation calculus induction axiom (2) is simply the induction principle for lists: If the empty list has property P and if, whenever list s has property P so does $cons(a, s)$, then all lists have property P.

[3] These foundational axioms are simpler than those presented by Reiter [33] and others. These earlier axiomatizations for the situation calculus are all derivable from the four axioms given here.

Example 2.1 When $f(\cdot,\cdot)$, $g(\cdot,\cdot)$ are functional fluents, $F(\cdot,\cdot,\cdot)$ is a relational fluent, and $P(\cdot,\cdot)$ is a non-fluent predicate, the following is uniform in σ:

$$(\forall x).x = f(g(x,\sigma),\sigma) \wedge (\exists y)F(g(A,\sigma),y,\sigma) \supset \neg P(x,B) \vee P(f(f(x,\sigma),\sigma),g(x,\sigma)).$$

No formula that mentions $Poss$ or $\sqsubset$ is uniform in any situation term σ. The following are not uniform in σ:

$holding(x, do(pickup(x),\sigma))$, $do(a,\sigma) \neq \sigma$, $(\exists s)holding(x,s)$,
$resigned(primeMinister(Canada, do(elect(p,\sigma)),\sigma))$.

Definition 2.2 Action Precondition Axiom
An action precondition axiom of $\mathcal{L}_{sitcalc}$ is a sentence of the form:

$$Poss(A(x_1,\ldots,x_n),s) \equiv \Pi_A(x_1,\ldots,x_n,s), \tag{5}$$

where A is an n-ary action function symbol, and $\Pi_A(x_1,\ldots,x_n,s)$ is a formula that is uniform in s and whose free variables are among $x_1,\ldots,x_n,s$.

For example, in a blocks world, we might typically have:

$Poss(pickup(x),s) \equiv (\forall y)\neg holding(y,s) \wedge \neg heavy(x,s)$.

The uniformity requirement on Π_A ensures that the preconditions for the executability of the action $A(x_1,\ldots,x_n)$ are determined only by the current situation s, not by any other situation.

Our decision to focus on sentences of the form (5) for axiomatizing action preconditions stems from the so-called *qualification problem* [23] and its nonmonotonic formulations. In those cases where closed-form solutions have been obtained to the associated circumscription policy (e.g. [20]), they have been biconditionals of the form (5).

Definition 2.3 Successor State Axiom

1. A successor state axiom for an $(n+1)$-ary relational fluent F is a sentence of $\mathcal{L}_{sitcalc}$ of the form:

$$F(x_1,\ldots,x_n,do(a,s)) \equiv \Phi_F(x_1,\ldots,x_n,a,s), \tag{6}$$

where $\Phi_F(x_1,\ldots,x_n,a,s)$ is a formula uniform in s, all of whose free variables are among $a,s,x_1,\ldots,x_n$. An example of such an axiom, taken from [31], is:

$$\begin{array}{l} broken(x,do(a,s)) \equiv \\ \quad (\exists r)\{a = drop(r,x) \wedge fragile(x,s)\} \vee (\exists b)\{a = explode(b) \wedge nextTo(b,x,s)\} \vee \\ \quad broken(x,s) \wedge \neg(\exists r)a = repair(r,x). \end{array}$$

This says that x will be broken in the successor situation $do(a,s)$ iff x was fragile in s and the action taking us to the successor situation was someone (r) dropping x, or the action was some bomb b exploding, and b was next to x, or x was already broken, and the action was not someone repairing x.

As for action precondition axioms, the uniformity of Φ_F guarantees that the truth value of $F(x_1, \ldots, x_n, do(a, s))$ in the successor situation $do(a, s)$ is determined entirely by the current situation s, and not by any other situation. In systems and control theory, this is often called the *Markov property.*

2. A successor state axiom for an $(n+1)$-ary functional fluent f is a sentence of $\mathcal{L}_{sitcalc}$ of the form:

$$f(x_1, \ldots, x_n, do(a, s)) = y \equiv \phi_f(x_1, \ldots, x_n, y, a, s),$$

where $\phi_f(x_1, \ldots, x_n, y, a, s)$ is a formula uniform in s, all of whose free variables are among $x_1, \ldots, x_n, y, a, s$. A blocks world example is:

$$\begin{aligned} &height(x, do(a, s)) = y \equiv a = moveToTable(x) \wedge y = 1 \ \vee \\ &\qquad (\exists z, h)(a = move(x, z) \wedge height(z, s) = h \wedge y = h + 1) \ \vee \\ &\qquad height(x, s) = y \wedge a \neq moveToTable(x) \wedge \neg(\exists z) a = move(x, z). \end{aligned}$$

As for relational fluents, the uniformity of ϕ_f in the successor state axioms for functional fluents guarantees the Markov property: The value of a functional fluent in a successor situation is determined entirely by properties of the current situation, and not by any other situation.

Following earlier ideas of Pednault [27], Haas [12], Schubert [37] and Davis [4], Reiter [31] shows how to solve the frame problem for deterministic actions.[4] The resulting solution yields axioms with exactly the syntactic form of successor state axioms, which is why in this paper we focus on these.

Basic Action Theories Henceforth, we shall consider theories $\mathcal{D}$ of $\mathcal{L}_{sitcalc}$ of the following forms:

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

where,

- Σ are the foundational axioms for situations.
- $\mathcal{D}_{ss}$ is a set of successor state axioms for functional and relational fluents, one for each such fluent of the language $\mathcal{L}_{sitcalc}$.
- $\mathcal{D}_{ap}$ is a set of action precondition axioms, one for each action function symbol of $\mathcal{L}_{sitcalc}$.
- $\mathcal{D}_{una}$ is the set of unique names axioms for all action function symbols of $\mathcal{L}_{sitcalc}$. For distinct action function symbols A and B of $\mathcal{L}_{sitcalc}$,

[4] This solution does not take ramification constraints into account, but see [20, 25, 28] for possible ways to do this, while preserving the successor state axiom approach.

$A(\vec{x}) \neq B(\vec{y})$.

Identical action terms have identical arguments:

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

- $\mathcal{D}_{S_0}$ is a set of first order sentences that are uniform in S_0. Thus, no sentence of $\mathcal{D}_{S_0}$ quantifies over situations, or mentions $Poss$, $\sqsubset$ or the function symbol do, so that S_0 is the only term of sort *situation* mentioned by these sentences. $\mathcal{D}_{S_0}$ will function as the initial theory of the world (i.e. the one we start off with, before any actions have been "executed"). Often, we shall call $\mathcal{D}_{S_0}$ the *initial database*. The initial database may (and often will) contain sentences mentioning no situation term at all, for example, unique names axioms for individuals, like $John \neq Mary$, or "timeless" facts like $isMountain(MtEverest)$, or $dog(x) \supset mammal(x)$.

Definition 2.4 A *basic action theory* is any collection of axioms $\mathcal{D}$ of the above form that also satisfies the following *functional fluent consistency property*:
Whenever f is a functional fluent whose successor state axiom in $\mathcal{D}_{ss}$ is

$$f(\vec{x}, do(a, s)) = y \equiv \phi_f(\vec{x}, y, a, s),$$

then

$$\mathcal{D}_{una} \cup \mathcal{D}_{S_0} \models (\forall a, s).(\forall \vec{x}).(\exists y)\phi_f(\vec{x}, y, a, s) \wedge \\ [(\forall y, y').\phi_f(\vec{x}, y, a, s) \wedge \phi_f(\vec{x}, y', a, s) \supset y = y'].$$

This consistency property provides a sufficient condition for preventing a source of inconsistency in f's successor state axiom. It says that the conditions defining f's value in the next situation $do(a, s)$, namely ϕ_f, actually define a value for f, and that this value is unique.

3 Relative Satisfiability of Basic Action Theories

We begin with a result that we shall need later, but that is of independent interest for the metatheory of the situation calculus.

Theorem 1 (Relative Satisfiability) *A basic action theory $\mathcal{D}$ is satisfiable iff $\mathcal{D}_{una} \cup \mathcal{D}_{S_0}$ is.*

This result assures us that provided the initial database together with the unique names axioms for actions are satisfiable, then unsatisfiability cannot be introduced by augmenting these with the foundational axioms for the situation calculus, together with action precondition and successor state axioms. In the absence of the above functional fluent consistency property, it is easy to construct examples of basic action theories for which this theorem is false.

Proof: Suppose $\mathcal{D}_{una} \cup \mathcal{D}_{S_0}$ has a model $\mathbb{M}_0$ with domains *Act* for the sort *action* and *Obj* for the sort *object*. Define a structure $\mathbb{M}$ as follows: $\mathbb{M}$'s domains for the sorts *action* and *object* are *Act* and *Obj* respectively. $\mathbb{M}$'s domain *Sit* for the sort *situation* is the set of all finite sequences of elements of *Act*. Next, we define how S_0 and *do* are interpreted by $\mathbb{M}$. In the following, we use $\xi^{\mathbb{M}}$ for the denotation of the symbol ξ in a structure $\mathbb{M}$.

1. $S_0^{\mathbb{M}} = [\,]$, the empty sequence.
2. Whenever $\alpha \in Act$ and $[\alpha_1, \ldots, \alpha_n] \in Sit$, $do^{\mathbb{M}}(\alpha, [\alpha_1, \ldots, \alpha_n])$ is defined to be the sequence $[\alpha_1, \ldots, \alpha_n]$ with the element α added to its end:

   $do^{\mathbb{M}}(\alpha, [\alpha_1, \ldots, \alpha_n]) = [\alpha_1, \ldots, \alpha_n, \alpha]$.
3. $\sigma \sqsubset^{\mathbb{M}} \sigma'$ iff the sequence σ is a proper initial subsequence of σ'.

While we have not yet finished specifying the structure $\mathbb{M}$, it is immediate that $\mathbb{M}$, as specified thus far, satisfies all the axioms of Σ:

1. When interpreted in $\mathbb{M}$, axiom (1) says that whenever two non-empty sequences are equal, their last elements are equal, and the sequences obtained by deleting their last elements are equal.
2. The induction axiom (2) simply says that *Sit* is the smallest set of sequences containing $[\,]$, and closed under the addition of an element of *Act* to the end of a sequence in the set.
3. Axiom (3) says that no sequence can be a proper initial subsequence of the empty sequence, and (4) says that $[\alpha_1, \ldots, \alpha_m]$ is a proper initial subsequence of $[\beta_1, \ldots, \beta_n, \alpha]$ iff it is an initial subsequence (not necessarily proper) of $[\beta_1, \ldots, \beta_n]$.

Next, we continue with the specification of $\mathbb{M}$ for interpreting predicate and function symbols other than *do*.

1. Since $\mathbb{M}$ and $\mathbb{M}_0$ share the same domains for sorts *object* and *action*, we can let $\mathbb{M}$ interpret situation independent predicates and functions exactly as does $\mathbb{M}_0$. It follows that, since $\mathbb{M}_0$ satisfies $\mathcal{D}_{una}$, so does $\mathbb{M}$.
2. Next, we specify how $\mathbb{M}$ interprets functional and relational fluents and *Poss* at S_0.

   (a) Let F be a relational fluent, and ν_0 a variable assignment for variables of sorts *object* and *action* for $\mathbb{M}_0$. Since $\mathbb{M}$ and $\mathbb{M}_0$ share the same domains of sorts *object* and *action*, it makes sense to define:

   $\mathbb{M}, \nu_0 \models F(\vec{x}, S_0)$ iff $\mathbb{M}_0, \nu_0 \models F(\vec{x}, S_0)$.

   When f is a functional fluent, define:

   $f^{\mathbb{M}}(\vec{x}[\nu_0], S_0^{\mathbb{M}}) = f^{\mathbb{M}_0}(\vec{x}[\nu_0], S_0^{\mathbb{M}_0})$.

   This is well defined because functional fluents take values of sort *action* or *object*, never of sort *situation*.

Now, all the sentences of $\mathcal{D}_{S_0}$ are uniform in S_0, which is to say, no such sentence mentions the predicates $Poss$ or $\sqsubset$, no such sentence quantifies over situations, and if such a sentence mentions a term of sort situation, that term is S_0. It follows that because $\mathbb{M}_0$ is a model of $\mathcal{D}_{S_0}$, then $\mathbb{M}$, as specified thus far, is also a model of $\mathcal{D}_{S_0}$.

(b) Now we specify how $\mathbb{M}$ interprets the predicate $Poss$ on S_0. Let $\alpha \in Act$. There are two possibilities:

(i) There is a variable assignment ν assigning α to the variable a, and an action function A of the language $\mathcal{L}_{sitcalc}$, such that $\mathbb{M}, \nu \models a = A(\vec{x})$. Now A must have an action precondition axiom of $\mathcal{D}_{ap}$ of the form $Poss(A(\vec{x}, s) \equiv \Pi_A(\vec{x}, s)$, where $\Pi_A(\vec{x}, s)$ is a formula that is uniform in s and whose free variables are among $\vec{x}, s$. Because $\Pi_A(\vec{x}, s)$ is uniform in s, $\Pi_A(\vec{x}, S_0)$ has already been assigned a truth value by $\mathbb{M}$ for every variable assignment. Now specify that

$$\mathbb{M}, \nu \models Poss(a, S_0) \text{ iff } \mathbb{M}, \nu \models \Pi_A(\vec{x}), S_0).$$

There remains the possibility that this definition does not specify a unique truth value for $Poss(a, S_0)$ under $\mathbb{M}$:

- Perhaps there is a variable assignment μ that is just like ν except that it assigns a different tuple of domain elements to $\vec{x}$, and $\mathbb{M}, \mu \models a = A(\vec{x})$. But this cannot happen because $\mathbb{M}$ satisfies the unique names axioms for actions.
- Perhaps there is a variable assignment μ assigning α to a, and an action function B of $\mathcal{L}_{sitcalc}$ different than A such that $\mathbb{M}, \mu \models a = B(\vec{y})$. But again, this cannot happen because $\mathbb{M}$ satisfies the unique names axioms for actions.

(ii) For every variable assignment ν assigning α to a and every action function A of $\mathcal{L}_{sitcalc}$, $\mathbb{M}, \nu \not\models a = A(\vec{x})$. In this case, whenever μ is a variable assignment assigning α to a, we are free to specify whether or not $\mathbb{M}, \mu \models Poss(a, S_0)$, so we just arbitrarily say it does.

Notice that this construction guarantees that every action precondition axiom is satisfied by $\mathbb{M}$ at S_0.

3. At this stage in its construction, $\mathbb{M}$ interprets all situation independent functions and predicates. Moreover, $\mathbb{M}$ interprets $Poss$ at S_0, and all functional and relational fluents at S_0, and it does so in such a way that the action precondition axioms are all satisfied at S_0. We now inductively extend this interpretation for $Poss$ and for fluents to situations other than S_0. So assume that $\mathbb{M}$ interprets $Poss$, and all functional and relational fluents at a situation s; we specify how $\mathbb{M}$ interprets these at situation $do(a, s)$. Moreover, we do so in such a way that the successor state and action precondition axioms will be satisfied.

Formally, suppose $\sigma \in Sit$, and that for every variable assignment ν that assigns σ to the variable s, $\mathbb{M}, \nu$ has interpreted $Poss(a, s)$, and $F(\vec{x}, s)$ for every n-ary relational fluent F, and that $\mathbb{M}, \nu$ has specified a value in $Obj \cup Act$ for $f(\vec{x}, s)$, for every n-ary functional fluent f.

(a) Suppose F is a relational fluent, with successor state axiom

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

For every variable assignment ν assigning σ to variable s, define:

$$\mathbb{M}, \nu \models F(\vec{x}, do(a, s)) \text{ iff } \mathbb{M}, \nu \models \Phi_F(\vec{x}, a, s).$$

This is well defined for two reasons:
(1) $\Phi_F(\vec{x}, a, s)$ is uniform in s, and therefore, has already been assigned a truth value in s by $\mathbb{M}$.
(2) The above truth assignments to $F(\vec{x}, do(a, s))$ cannot conflict with any truth assignments made to F at an earlier step of the construction of $\mathbb{M}$ because those earlier truth assignments were made in situations different than $do(a, s)$.

It follows from this definition that $\mathbb{M}$ now satisfies F's successor state axiom at $do(a, s)$.

(b) Suppose f is a functional fluent, with successor state axiom

$$f(\vec{x}, do(a, s)) = y \equiv \phi_f(\vec{x}, y, a, s).$$

Let ν be a variable assignment assigning σ to variable s. Then define:

$$\mathbb{M}, \nu \models f(\vec{x}, do(a, s)) = y \text{ iff } \mathbb{M}, \nu \models \Phi_f(\vec{x}, y, a, s). \tag{7}$$

This is well defined for three reasons:
(1) $\Phi_f(\vec{x}, y, a, s)$ is uniform in s, and therefore, has already been assigned a truth value in s by $\mathbb{M}$.
(2) The above value assignments to $f(\vec{x}, do(a, s))$ cannot conflict with any value assignments made to f at an earlier step of the construction of $\mathbb{M}$ because those earlier value assignments were made in situations different than $do(a, s)$.
(3) The functional fluent consistency property states that:

$$\begin{aligned} \mathcal{D}_{una} \cup \mathcal{D}_{S_0} \models (\forall a, s, \vec{x}).(\exists y)\phi_f(\vec{x}, y, a, s) \wedge \\ [(\forall y, y').\phi_f(\vec{x}, y, a, s) \wedge \phi_f(\vec{x}, y', a, s) \supset y = y']. \end{aligned}$$

Since by construction, $\mathbb{M}$ satisfies $\mathcal{D}_{una} \cup \mathcal{D}_{S_0}$, it follows that

$$\begin{aligned} \mathbb{M}, \nu \models (\forall \vec{x}).(\exists y)\phi_f(\vec{x}, y, a, s) \wedge \\ [(\forall y, y').\phi_f(\vec{x}, y, a, s) \wedge \phi_f(\vec{x}, y', a, s) \supset y = y']. \end{aligned}$$

In other words, (7) assigns one and only one value y to $f(\vec{x}, do(a, s))$.

It follows from this definition that $\mathbb{M}$ now satisfies f's successor state axiom at $do(a, s)$.

(c) Finally, we need to specify how $\mathbb{M}, \nu$ interprets $Poss(a', do(a,s))$, for every variable assignment ν that assigns σ to variable s. The construction for this is exactly parallel to that of case 2b above for S_0, using the fact that $\mathbb{M}, \nu$ now interprets all fluents at $do(a,s)$. A consequence of this construction is that $\mathbb{M}$ will satisfy all of the action precondition axioms of $\mathcal{D}_{ap}$ at $do(a,s)$.

This completes the construction of $\mathbb{M}$'s interpretation for $Poss$ and fluents. By the the nature of this construction, $\mathbb{M}$ satisfies all the action precondition and successor state axioms of $\mathcal{D}$.

■

4 Regression

Regression [40] is a central computational mechanism that arises again and again in artificial intelligence applications, forming the basis for many planning procedures and for automated reasoning in the situation calculus. Typically, regression is used for addressing the so-called *projection problem.* Roughly speaking, this is the problem of determining whether a sentence is true of one or more "reasonably specified" future situations. For example, in a blocks world, one might wish to know whether there are two different blocks such that it is possible to move one to the table then move the other onto A, and whether after that, all blocks other than A will be clear: Determine whether

$$\begin{aligned}\mathcal{D} \models (\exists x, y).&Poss(moveToTable(x), S_0) \wedge \\ &Poss(move(y, A), do(moveToTable(x), S_0)) \wedge x \neq y \wedge \\ &(\forall z).z \neq A \supset clear(z, do(move(y, A), do(moveToTable(x), S_0))).\end{aligned} \tag{8}$$

Or one might wish to know whether there is a sequence of two actions that leads to a state containing a three block tower whose top block is that block that was initially the bottom of tower A: Determine whether

$$\begin{aligned}\mathcal{D} \models (\exists a, a', x, y).&ontable(x, do(a, do(a', S_0))) \wedge on(y, x, do(a, do(a', S_0))) \wedge \\ &on(bottom(A, S_0), y, do(a, do(a', S_0))) \wedge \\ &clear(bottom(A, S_0), do(a, do(a', S_0))).\end{aligned} \tag{9}$$

Our purpose in this section is first to formally characterize a general notion of "reasonably specified future situation" (via the *regressable* formulas defined below), next to define a regression operator for the purposes of solving projection problems like those just given, finally to prove the soundness and completeness of this method.

The intuition underlying regression is this: Suppose we want to prove that a sentence W is entailed by some basic action theory. Suppose further that W mentions a relational fluent atom $F(\vec{t}, do(\alpha, \sigma))$, where F's successor state axiom is $F(\vec{x}, do(a,s)) \equiv \Phi_F(\vec{x}, a, s)$. Then we can easily determine a logically equivalent sentence W' by substituting $\Phi_F(\vec{t}, \alpha, \sigma)$ for

$F(\vec{t}, do(\alpha, \sigma))$ in W. After we do so, the fluent atom $F(\vec{t}, do(\alpha, \sigma))$, involving the complex situation term $do(\alpha, \sigma)$, has been eliminated from W in favour of $\Phi_F(\vec{t}, \alpha, \sigma)$, and this involves the simpler situation term σ. In this sense, W' is "closer" to the initial situation S_0 than was W. Moreover, this operation can be repeated until the resulting goal formula mentions only the situation term S_0, after which, intuitively, it should be sufficient to establish this resulting goal using only the sentences of the initial database. Regression is a mechanism that repeatedly performs the above reduction starting with a goal W, ultimately obtaining a logically equivalent goal W_0 whose only situation term is S_0. We have only indicated how regression works by reducing relational fluent atoms in W; there is an analogous way of reducing functional fluent terms.

Notational Convention 4.1 Let $\alpha_1, \ldots, \alpha_n$ be terms of sort *action*, and σ a term of sort *situation*. Define

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

$$do([\alpha_1, \ldots, \alpha_n], \sigma) = do(\alpha_n, do([\alpha_1, \ldots, \alpha_{n-1}], \sigma)) \quad n = 1, 2, \ldots$$

$do([\alpha_1, \ldots, \alpha_n], \sigma)$ is a convenient notation for the situation term $do(\alpha_n, do(\alpha_{n-1}, \ldots do(\alpha_1, \sigma) \ldots))$, denoting that situation resulting from performing the action α_1, followed by $\alpha_2, \ldots$, followed by α_n, beginning in situation σ.

Definition 4.1 The Regressable Formulas. A formula W of $\mathcal{L}_{sitcalc}$ is *regressable* iff

1. W is first order.
2. Every term of sort *situation* mentioned by W has the form $do([\alpha_1, \ldots, \alpha_n], S_0)$ for some $n \geq 0$, and for terms $\alpha_1, \ldots, \alpha_n$ of sort *action*.
3. For every atom of the form $Poss(\alpha, \sigma)$ mentioned by W, α has the form $A(t_1, \ldots, t_n)$ for some n-ary action function symbol A of $\mathcal{L}_{sitcalc}$.
4. W does not quantify over situations.[5]

Example 4.1 The sentences of (8) and (9) are regressable. These remain regressable if one or more of their quantifiers are deleted. In other words, we treat *formulas* as well as sentences. The availability of functional fluents allows quite complex (perhaps strange) situations and regressable formulas to be represented. For example,

$$\begin{aligned} &happy(Mary, do(pickup(Mary, favoriteBook(John, do(read(John, MobyDick), S_0))), \\ &\qquad do(walkTo(John, houseOf(Mary, do(divorce(Mary, John), S_0))), \\ &\qquad\quad do(marry(Mary, John), S_0)))) \end{aligned}$$

[5]Strictly speaking, this condition is of no real consequence. Because of condition 2, the variable s of quantification in $(\forall s)W$ cannot be mentioned in W, so the quantifier is doing no real work. But this no-quantification condition will slightly simplify the analysis to follow.

claims that Mary would be happy as a result of the following sequence of actions: First, Mary marries John, then John walks to the house that Mary would have owned had she initially divorced John, then Mary picks up what John's favorite book would be had he initially read Moby Dick. The following are not regressable:

$(\exists a)Poss(a, S_0), \quad holding(x, do(pickup(A), s)).$

The essence of a regressable formula is that each of its *situation* terms is rooted at S_0, and therefore, one can tell, by inspection of such a term, exactly how many actions it involves. It is not necessary to be able to tell what those actions are, just how many there are. In addition, when a regressable formula mentions a *Poss* atom, we can tell, by inspection of that atom, exactly what is the action function symbol occurring in its first argument position, for example, that it is a *move* action.

For the purposes of defining the regression operator, we require the following:

Definition 4.2 Prime functional fluent terms
A functional fluent term is *prime* iff it has the syntactic form $f(\vec{t}, do([\alpha_1, \ldots, \alpha_n], S_0))$ for $n \geq 1$ and each of the terms $\vec{t}, \alpha_1, \ldots, \alpha_n$ is uniform in S_0. Thus, for prime functional fluent terms, S_0 is the only term of sort *situation* (if any) mentioned by $\vec{t}, \alpha_1, \ldots, \alpha_n$.

Remark 1 *Suppose that $g(\vec{\tau}, do(\alpha, \sigma))$ has the property that every term of sort* situation *that it mentions has the form $do([\alpha_1, \ldots, \alpha_n], S_0)$ for some $n \geq 0$. Then $g(\vec{\tau}, do(\alpha, \sigma))$ mentions a prime functional fluent term.*

Proof: For $n \geq 0$, define the *length* of the situation term $do([a_1, \ldots, a_n], S_0)$ to be n. Now use induction on the sum of the lengths of all terms of sort *situation* mentioned by $g(\vec{\tau}, do(\alpha, \sigma))$ (with base case 1). ■

Example 4.2 Suppose $f(\cdot, \cdot, \cdot)$, $r(\cdot)$ and $h(\cdot)$ are functional fluent symbols, and $g(\cdot, \cdot)$ is an ordinary function symbol. Then the functional fluent term

$$f(g(h(do(A, S_0)), h(S_0)), f(B, r(S_0), do(A, do(B, S_0))), do(h(do(C, S_0)), S_0))$$

mentions three prime functional fluent terms:

$h(do(A, S_0))$,

$f(B, r(S_0), do(A, do(B, S_0)))$ and

$h(do(C, S_0))$.

Following Pednault [27], we now define the regression operator.[6]

[6] For the purposes of proving the soundness and completeness of regression (Theorem 3 below), our definition differs in several significant ways from Pednault's. In particular, we introduce the concept of a prime functional fluent term to guarantee that the regression operator is well defined. Pednault himself proves neither soundness nor completeness.

Definition 4.3 The Regression Operator.
The *regression operator* $\mathcal{R}$ when applied to a regressable formula W of $\mathcal{L}_{sitcalc}$ is determined relative to a basic theory of actions of $\mathcal{L}_{sitcalc}$ that serves as a background axiomatization. As we shall prove, when applied to a regressable sentence, the regression operator produces a logically equivalent sentence whose only situation term is S_0. In what follows, $\vec{t}$, $\vec{\tau}$ are tuples of terms; α, with or without subscripts and superscripts, is a term of sort *action*; σ, σ' are terms of sort *situation*; and W is a regressable formula of $\mathcal{L}_{sitcalc}$. We emphasize here that *the regression operator is defined only for regressable formulas of* $\mathcal{L}_{sitcalc}$.

1. Suppose that W is a regressable equality atom between two *situation* terms, so it has the form:

   $$do([\alpha_1, \ldots, \alpha_m], S_0) = do([\alpha'_1, \ldots, \alpha'_n], S_0).$$

   If $m = 0$ (so $do([\alpha_1, \ldots, \alpha_m], S_0)$ is simply S_0), and if $n = 0$, then

   $$\mathcal{R}[W] = true.$$

   If $m \neq n$, then

   $$\mathcal{R}[W] = false.$$

   If $m = n \geq 1$, then

   $$\mathcal{R}[W] = \mathcal{R}[\alpha_1 = \alpha'_1 \wedge \cdots \wedge \alpha_m = \alpha'_m].$$

2. Suppose that W is a regressable $\sqsubset$ atom, so it has the form:

   $$do([\alpha_1, \ldots, \alpha_m], S_0) \sqsubset do([\alpha'_1, \ldots, \alpha'_n], S_0).$$

   If $m = 0$ and $n \geq 1$, then

   $$\mathcal{R}[W] = true.$$

   If $m \geq n$, then

   $$\mathcal{R}[W] = false.$$

   If $1 \leq m < n$, then

   $$\mathcal{R}[W] = \mathcal{R}[\alpha_1 = \alpha'_1 \wedge \cdots \wedge \alpha_m = \alpha'_m].$$

3. Suppose W is a regressable *Poss* atom, so it has the form $Poss(A(\vec{t}), \sigma)$ for terms $A(\vec{t})$ and σ of sort *action* and *situation* respectively. Here, A is an action function symbol of $\mathcal{L}_{sitcalc}$. Then there must be an action precondition axiom for A of the form

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

   Without loss of generality, assume that all quantifiers (if any) of $\Pi_A(\vec{x}, s)$ have had their quantified variables renamed to be distinct from the free variables (if any) of $Poss(A(\vec{t}), \sigma)$. Then

   $$\mathcal{R}[W] = \mathcal{R}[\Pi_A(\vec{t}, \sigma)].$$

   In other words, replace the atom $Poss(A(\vec{t}), \sigma)$ by its definition, and regress that expression. The above renaming of quantified variables of $\Pi_A F(\vec{x}, a, s)$ prevents any of these quantifiers from capturing variables in the instance $Poss(A(\vec{t}), \sigma)$.

4. For the remaining possibilities when W is a regressable atom:

   (a) If S_0 is the only term of sort *situation* (if any) mentioned by W, then

   $\mathcal{R}[W] = W.$

   (b) Suppose that W mentions a term of the form $g(\vec{\tau}, do(\alpha', \sigma'))$ for some functional fluent g. Then by the fact that W is regressable and Remark 1, $g(\vec{\tau}, do(\alpha', \sigma'))$ mentions a prime functional fluent term. Let this prime term have the form $f(\vec{t}, do(\alpha, \sigma))$, and suppose f's successor state axiom in $\mathcal{D}_{ss}$ is

   $f(\vec{x}, do(a, s)) = y \equiv \phi_f(\vec{x}, y, a, s).$

   Without loss of generality, assume that all quantifiers (if any) of $\phi_f(\vec{x}, y, a, s)$ have had their quantified variables renamed to be distinct from the free variables (if any) of $f(\vec{t}, do(\alpha, \sigma))$. Then

   $\mathcal{R}[W] = \mathcal{R}[(\exists y).\phi_f(\vec{t}, y, \alpha, \sigma) \wedge W|^{f(\vec{t}, do(\alpha,\sigma))}_{y}].$[7]

   Here, y is a variable not occurring free in $W, \vec{t}, \alpha$ or σ. Notice that, relative to f's successor state axiom, we have simply replaced W by a logically equivalent formula $(\exists y).\phi_f(\vec{t}, y, \alpha, \sigma) \wedge W|^{f(\vec{t}, do(\alpha,\sigma))}_{y}$, and we next regress this formula.[8] The above renaming of quantified variables of $\phi_f(\vec{x}, y, a, s)$ prevents any of these quantifiers from capturing variables in the instance $f(\vec{t}, do(\alpha, \sigma))$.

   (c) The remaining possibility is that W is a relational fluent atom of the form $F(\vec{t}, do(\alpha, \sigma))$, and moreover, W does not mention a functional fluent term of the form $g(\vec{\tau}, do(\alpha', \sigma'))$. Let F's successor state axiom in $\mathcal{D}_{ss}$ be

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

   Without loss of generality, assume that all quantifiers (if any) of $\Phi_F(\vec{x}, a, s)$ have had their quantified variables renamed to be distinct from the free variables (if any) of $F(\vec{t}, do(\alpha, \sigma))$. Then

   $\mathcal{R}[W] = \mathcal{R}[\Phi_F(\vec{t}, \alpha, \sigma)].$

   In other words, replace W by a suitable instance of the right hand side of the equivalence in F's successor state axiom, and regress this formula. The above renaming of quantified variables of $\Phi_F(\vec{x}, a, s)$ prevents any of these quantifiers from capturing variables in the instance $F(\vec{t}, do(\alpha, \sigma))$.

[7] In general, when ϕ is a formula, and t and t' are terms, then $\phi|^{t'}_{t}$ denotes that formula obtained from ϕ by replacing all occurrences of t' in ϕ by t.

[8] Notice that, strictly speaking, $\mathcal{R}$ does not really define a function in this case, because W might mention two (or more) prime functional fluent terms suitable for the above regression step (e.g. Example 4.2), and we haven't specified which of these is to be used. So, to be really precise, we can suppose a suitable lexicographic ordering on functional fluent terms, and regress with respect to the least such prime term mentioned by W. In view of our principal results about regression, proved below, none of this really matters.

5. For non-atomic formulas, regression is defined inductively.

$$\mathcal{R}[\neg W] = \neg\mathcal{R}[W],$$
$$\mathcal{R}[W_1 \wedge W_2] = \mathcal{R}[W_1] \wedge \mathcal{R}[W_2],$$
$$\mathcal{R}[(\exists v)W] = (\exists v)\mathcal{R}[W].$$

As described above, regression is defined only for regressable formulas of $\mathcal{L}_{sitcalc}$. For example, $\mathcal{R}[Poss(a, S_0)]$ is undefined, as is $\mathcal{R}[holding(x, do(pickup(A), s))]$, because in both cases, the regression operator is being applied to non-regressable formulas.

The idea behind the regression operator $\mathcal{R}$ is to progressively reduce the length of the situation terms of a regressable formula W by eliminating terms of the form $do(\alpha, \sigma)$ mentioned in W in favour of suitable instances of Φ_F and ϕ_f. Since Φ_F and ϕ_f are uniform in s, they do not mention the function symbol do, so the effect of a single step of regression is to replace W by a formula with one less total depth of nesting of the function symbol do. Therefore, intuitively, the final result of regressing W should be a formula whose only situation term is S_0. This is precisely the notion of *goal regression* in artificial intelligence planning procedures, although there, functional fluents are not normally considered (but see Pednault [26] for an exception).

That the regression operator is well defined is by no means obvious; the following theorem confirms that it is, and also that regressing W produces an equivalent formula about the initial situation S_0 only.

Theorem 2 *Suppose W is a regressable formula of $\mathcal{L}_{sitcalc}$ and $\mathcal{D}$ is a basic theory of actions. Then $\mathcal{R}[W]$ is a formula uniform in S_0. Moreover,*

$$\mathcal{D} \models (\forall)(W \equiv \mathcal{R}[W]),$$

where $(\forall)\phi$ denotes the universal closure of the formula ϕ with respect to its free variables.

Proof: For the purposes of giving an inductive proof of this theorem, we require a suitable well-founded ordering relation, which we now define.

Consider the set Λ of all countably infinite sequences of natural numbers with a finite number of non-zero elements, and the following binary relation $\prec$ (the *reverse lexicographic order*) on this set:

$$(\lambda_1, \lambda_2, \ldots) \prec (\lambda'_1, \lambda'_2, \ldots) \text{ iff for some } m,\ \lambda_m < \lambda'_m, \text{ and for all } n > m,\ \lambda_n = \lambda'_n.$$

$(\Lambda, \prec)$ is well founded, with minimal element $(0, 0, \ldots)$. Next, let $\mathbf{3}$ be the set of all 3-tuples of natural numbers. We overload the relation $\prec$ by defining a reverse lexicographic ordering on $\mathbf{3}$:

$$(m_1, m_2, m_3) \prec (n_1, n_2, n_3) \text{ iff } m_3 < n_3 \text{ or } m_3 = n_3 \text{ and } m_2 < n_2 \text{ or } m_3 = n_3 \text{ and } m_2 = n_2 \text{ and } m_1 < n_1.$$

Finally (again overloading $\prec$), define an ordering on $\Lambda\times\mathbf{3}$ by:

$$(\lambda, f) \prec (\lambda', f') \text{ iff } f \prec f' \text{ or } f = f' \text{ and } \lambda \prec \lambda'.$$

The relation $\prec$ on $\Lambda \times \mathbf{3}$ is well founded, with minimal element $((0,0,\ldots),(0,0,0))$, and therefore can serve as a basis for an inductive proof.

For $n \geq 0$, define the *length* of the situation term $do([a_1,\ldots,a_n],S_0)$ to be n. Whenever $g(t_1,\ldots,t_n)$ is a term of $\mathcal{L}_{sitcalc}$, $t_1,\ldots,t_n$ are said to be *proper subterms* of $g(t_1,\ldots,t_n)$. An occurrence of a situation term in a formula W of $\mathcal{L}_{sitcalc}$ is *maximal* iff its occurrence is not as a proper subterm of some situation term. Given a regressable formula W, define

$$index(W) = ((C,\lambda_1,\lambda_2,\ldots),(P,E,I)),$$

where:

1. C is the total number of logical connectives and quantifiers mentioned by W.
2. For $m \geq 1$, λ_m is the number of occurrences in W of maximal situation terms of length m.
3. P is the number of atoms of the form $Poss(\alpha,\sigma)$ mentioned by W, E is the number of atoms of the form $\sigma_1 = \sigma_2$ mentioned by W, and I is the number of atoms of the form $\sigma_1 \sqsubset \sigma_2$ mentioned by W, where σ_1, σ_2 are terms of sort *situation*.

The proof of the theorem is by induction on the index of W relative to the ordering $\prec$.

When the index is $((0,0,\ldots),(0,0,0))$, W must be an atom that does not mention $Poss$, $\sqsubset$ or equality between situations, and the only situation term (if any) mentioned by W is S_0. In other words, W is uniform in S_0. Moreover, by definition of the regression operator, $\mathcal{R}[W] = W$ and the theorem is immediate.

Suppose $index(W) \succ ((0,0,\ldots),(0,0,0))$, and assume the theorem for all regressable formulas of index $\prec \nu$.

1. Suppose that W is a regressable equality atom between two *situation* terms, so it has the form:

   $$do([\alpha_1,\ldots,\alpha_m],S_0) = do([\alpha'_1,\ldots,\alpha'_n],S_0).$$

   (a) If $m = n = 0$, then W is $S_0 = S_0$, $\mathcal{R}[W]$ is $true$, and the theorem is immediate.

   (b) If $m \neq n$, then $\mathcal{R}[W]$ is $false$; moreover, $\Sigma \models (\forall).W \equiv false$. Therefore, the theorem is immediate.

   (c) If $m = n \geq 1$, then

   $$\Sigma \models (\forall).W \equiv \alpha_1 = \alpha'_1 \wedge \cdots \wedge \alpha_m = \alpha'_m. \tag{10}$$

   Moreover, $\alpha_1 = \alpha'_1 \wedge \cdots \wedge \alpha_m = \alpha'_m$ is regressable, and has index $\prec \nu$. Therefore, by induction hypothesis, $\mathcal{R}[\alpha_1 = \alpha'_1 \wedge \cdots \wedge \alpha_m = \alpha'_m]$ is uniform in S_0, and

   $$\mathcal{D} \models (\forall).\alpha_1 = \alpha'_1 \wedge \cdots \wedge \alpha_m = \alpha'_m \equiv \mathcal{R}[\alpha_1 = \alpha'_1 \wedge \cdots \wedge \alpha_m = \alpha'_m].$$

   The theorem follows from this and (10).

2. Suppose that W is a regressable atom of the form:

   $do([\alpha_1, \ldots, \alpha_m], S_0) \sqsubset do([\alpha'_1, \ldots, \alpha'_n], S_0)$.

   (a) If $m = 0$ and $n \geq 1$, then $\mathcal{R}[W] = true$. Moreover, $\Sigma \models (\forall).W \equiv true$. The theorem is now immediate.

   (b) If $m \geq n$, then $\mathcal{R}[W] = false$. Moreover, $\Sigma \models (\forall).W \equiv false$. The theorem is now immediate.

   (c) If $1 \leq m < n$, the theorem follows by exactly the same argument as in 1c above.

3. Suppose W is a regressable atom of the form $Poss(A(\vec{t}), \sigma)$ for terms $A(\vec{t})$ and σ of sort *action* and *situation* respectively. Here, A is an action function symbol of $\mathcal{L}_{sitcalc}$. Then there must be an action precondition axiom in $\mathcal{D}$ for A of the form

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

   Here, all quantifiers (if any) of $\Pi_A(\vec{x}, s)$ have had their quantified variables renamed to be distinct from all of the free variables (if any) of $Poss(A(\vec{t}), \sigma)$. Then by definition,

   $$\mathcal{R}[W] = \mathcal{R}[\Pi_A(\vec{t}, \sigma)], \tag{11}$$

   and by the variable renaming assumption,

   $$\mathcal{D} \models (\forall).W \equiv \Pi_A(\vec{t}, \sigma). \tag{12}$$

   Now $\Pi_A(\vec{x}, s)$ is uniform in s, so in particular, it does not mention the predicate symbols $Poss$ or $\sqsubset$, nor does it mention any equality atoms between situations. Therefore, $index(\Pi_A(\vec{t}, \sigma)) \prec index(W)$. Moreover, because $Poss(A(\vec{t}), \sigma)$ is regressable, and because $\Pi_A(\vec{x}, s)$ is uniform in s, $\Pi_A(\vec{t}, \sigma)$ is regressable, so by induction hypothesis, $\mathcal{R}[\Pi_A(\vec{t}, \sigma)]$ is a formula uniform in S_0, and

   $$\mathcal{D} \models (\forall).\Pi_A(\vec{t}, \sigma) \equiv \mathcal{R}[\Pi_A(\vec{t}, \sigma)]. \tag{13}$$

   The theorem now follows from (11), (12) and (13).

4. For the remaining possibilities when W is a regressable atom:

   (a) The possibility that S_0 is the only situation term (if any) mentioned by W is precluded by the assumption that $index(W) \succ ((0, 0, \ldots), (0, 0, 0))$.

   (b) Suppose that W mentions a functional fluent term of the form $g(\vec{\tau}, do(\alpha', \sigma'))$. Therefore, by the fact that W is regressable and Remark 1, $g(\vec{\tau}, do(\alpha', \sigma'))$ mentions a prime functional fluent term. Let this prime term have the form $f(\vec{t}, do(\alpha, \sigma))$, and suppose f's successor state axiom in $\mathcal{D}_{ss}$ is

   $f(\vec{x}, do(a, s)) = y \equiv \phi_f(\vec{x}, y, a, s)$.

   Here, all quantifiers (if any) of $\phi_f(\vec{x}, y, a, s)$ have had their quantified variables renamed to be distinct from the free variables (if any) of $f(\vec{t}, do(\alpha,\sigma))$. Then by definition

$$\mathcal{R}[W] = \mathcal{R}[(\exists y).\phi_f(\vec{t}, y, \alpha, \sigma) \wedge W|^{f(\vec{t}, do(\alpha,\sigma))}_y]. \tag{14}$$

Here, y is a variable not occurring free in $W, \vec{t}, \alpha$ or σ. By the above variable renaming assumption,

$$\mathcal{D} \models (\forall).W \equiv (\exists y).\phi_f(\vec{t}, y, \alpha, \sigma) \wedge W|^{f(\vec{t}, do(\alpha,\sigma))}_y \tag{15}$$

Now $\phi_f(\vec{x}, y, a, s)$ is uniform in s, so in particular, it does not mention the predicate symbols $Poss$ or $\sqsubset$, nor does it mention any equality atoms between situations, nor does it mention the function symbol do. Moreover, $f(\vec{t}, do(\alpha,\sigma))$ is a prime functional fluent term. Therefore,

$$index((\exists y).\phi_f(\vec{t}, y, \alpha, \sigma) \wedge W|^{f(\vec{t}, do(\alpha,\sigma))}_y) \prec index(W).$$

Also,

$$(\exists y).\phi_f(\vec{t}, y, \alpha, \sigma) \wedge W|^{f(\vec{t}, do(\alpha,\sigma))}_y$$

is regressable because W is, and because $\phi_f(\vec{x}, y, a, s)$ is uniform in s, so by induction hypothesis,

$$\mathcal{R}[(\exists y).\phi_f(\vec{t}, y, \alpha, \sigma) \wedge W|^{f(\vec{t}, do(\alpha,\sigma))}_y]$$

is a formula uniform in S_0, and

$$\begin{aligned}\mathcal{D} \models (\forall).(\exists y)\{\phi_f(\vec{t}, y, \alpha, \sigma) \wedge W|^{f(\vec{t}, do(\alpha,\sigma))}_y\} \equiv \\ \mathcal{R}[(\exists y)\{\phi_f(\vec{t}, y, \alpha, \sigma) \wedge W|^{f(\vec{t}, do(\alpha,\sigma))}_y\}].\end{aligned} \tag{16}$$

The theorem now follows from (14), (15) and (16).

(c) The remaining possibility is that W is a relational fluent atom of the form $F(\vec{t}, do(\alpha,\sigma))$, and moreover, W does not mention a functional fluent term of the form $g(\vec{\tau}, do(\alpha',\sigma'))$. Let F's successor state axiom in $\mathcal{D}_{ss}$ be

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

Here, all quantifiers (if any) of $\Phi_F(\vec{x}, a, s)$ have had their quantified variables renamed to be distinct from any of the free variables (if any) of $F(\vec{t}, \sigma)$. Then by definition,

$$\mathcal{R}[W] = \mathcal{R}[\Phi_F(\vec{t}, \alpha, \sigma)], \tag{17}$$

and by the variable renaming assumption,

$$\mathcal{D} \models (\forall).W \equiv \Phi_F(\vec{t}, \alpha, \sigma). \tag{18}$$

Now $\Phi_F(\vec{x}, a, s)$ is uniform in s, so in particular, it does not mention the predicate symbols $Poss$ or $\sqsubset$, nor does it mention any equality atoms between situations, nor does it mention the function symbol do. Moreover, by assumption,

$F(\vec{t}, do(\alpha, \sigma))$ does not mention a functional fluent term of the form $g(\vec{\tau}, do(\alpha', \sigma'))$. Therefore,

$$index(\Phi_F(\vec{t}, \alpha, \sigma)) \prec index(W).$$

Also, $\Phi_F(\vec{t}, \alpha, \sigma)$ is regressable because $F(\vec{t}, do(\alpha, \sigma))$ is, and because $\phi_F(\vec{x}, a, s)$ is uniform in s, so by induction hypothesis, $\mathcal{R}[\Phi_F(\vec{t}, \alpha, \sigma)]$ is a formula uniform in S_0, and

$$\mathcal{D} \models (\forall).\Phi_F(\vec{t}, \alpha, \sigma) \equiv \mathcal{R}[\Phi_F(\vec{t}, \alpha, \sigma)]. \tag{19}$$

The theorem now follows from (17), (18) and (19).

5. The rest of the proof is straightforward when W is not an atomic formula.

■

Lemma 4.1 *Suppose that $\mathcal{D}$ is a basic theory of actions of $\mathcal{L}_{sitcalc}$. If ϕ is a sentence of $\mathcal{L}_{sitcalc}$ that is uniform in S_0, then $\mathcal{D} \models \phi$ iff $\mathcal{D}_{S_0} \cup \mathcal{D}_{una} \models \phi$.*

Proof: The $\Leftarrow$ direction is trivial. For the other direction, suppose that $\mathcal{D} \models \phi$. Then $\mathcal{D} \cup \{\neg\phi\}$ is unsatisfiable. Now, $\mathcal{D}_{S_0} \cup \{\neg\phi\}$ has the syntactic form required of an initial database. Therefore, if $\mathcal{D} = \Sigma \cup \mathcal{D}_{ss} \cup \mathcal{D}_{ap} \cup \mathcal{D}_{una} \cup \mathcal{D}_{S_0}$, then
$\Sigma \cup \mathcal{D}_{ss} \cup \mathcal{D}_{ap} \cup \mathcal{D}_{una} \cup (\mathcal{D}_{S_0} \cup \{\neg\phi\})$ is a basic action theory.
By Theorem 1, $\mathcal{D}_{S_0} \cup \{\neg\phi\} \cup \mathcal{D}_{una}$ is unsatisfiable, from which the conclusion is immediate. ■

As an immediate consequence of this lemma and the preceding theorem, we have:

Theorem 3 (Soundness and Completeness of Regression)
Suppose W is a regressable sentence of $\mathcal{L}_{sitcalc}$ and $\mathcal{D}$ is a basic theory of actions. Then,

1. *$\mathcal{R}[W]$ is a sentence uniform in S_0.*
2. *$\mathcal{D} \models W$ iff $\mathcal{D}_{S_0} \cup \mathcal{D}_{una} \models \mathcal{R}[W]$.*

This is the main result of this section. It shows how, using regression, one can reduce the problem of proving a regressable sentence in a basic action theory to an entailment problem relative to the initial database and the unique names axioms for actions. The foundational axioms Σ of the situation calculus are not required for establishing this entailment. The action precondition and successor state axioms are also not required; their effects have been compiled into the regressed formula.

From a computational perspective, regression can be problematic. It is easy to construct examples of successor state axioms that lead to regressed formulas whose lengths are exponential in the number of actions. Fortunately, for a wide class of successor state axioms, one can prove good complexity results – linear in the number of actions – for regression [30], but pursuing such questions here would take us too far afield.

5 Σ-Reduction

For the purposes of automating deduction in the situation calculus, the foundational axioms Σ are problematic, especially the induction axiom. Accordingly, it would be desirable to characterize broad classes of sentences whose proofs need never appeal to induction, or even better, can additionally ignore some of the other axioms of Σ, or best of all, need never appeal to any of the foundational axioms Σ. That is the objective of this section. Specifically, we show that in proving sentences whose prenex forms involve only existentially quantified *situation* variables, induction is never needed. Moreover, under suitable additional conditions on such sentences, certain other axioms of Σ are also not needed. In proving results of this kind, the following definition will play a central role.

Definition 5.1 Principal Substructure.
Suppose $\mathbb{S}$ is a structure for $\mathcal{L}_{sitcalc}$. Its *principal substructure*, $\mathbb{S}_{pr}$, is that substructure of $\mathbb{S}$ with the same *object* domain as $\mathbb{S}$, the same *action* domain $\mathcal{A}$ as $\mathbb{S}$, and with *situation* domain:

$$\{S_0^{\mathbb{S}}\} \cup \{do^{\mathbb{S}}(\alpha_1, do^{\mathbb{S}}(\alpha_2, \ldots, do^{\mathbb{S}}(\alpha_n, S_0^{\mathbb{S}}) \cdots)) \mid n \geq 1,\ \alpha_1, \ldots, \alpha_n \in \mathcal{A}\}.^9$$

Lemma 5.1 *Suppose that ϕ is a first order formula of $\mathcal{L}_{sitcalc}$. of the form:*

$$\vec{Q}_1(\exists s_1)\vec{Q}_2(\exists s_2)\cdots(\exists s_n)\vec{Q}_n\psi,$$

where $s_1, \ldots, s_n$ are variables of sort situation*, ψ is quantifier free, and each $\vec{Q}_i$ is a sequence of zero or more quantifiers over variables of sort action $\cup$ object. Suppose further that $\mathbb{S}$ is a structure for $\mathcal{L}_{sitcalc}$. Then, whenever ν is an assignment to the variables of $\mathcal{L}_{sitcalc}$ of values from the domain of $\mathbb{S}_{pr}$,*

$$\mathbb{S}_{pr}, \nu \models \phi \ \ \Rightarrow \ \ \mathbb{S}, \nu \models \phi.$$

Proof: By induction on the number of quantifiers mentioned by ϕ. When there are no quantifiers, the result follows by a simple induction on the syntactic structure of ϕ. Assume the result when ϕ mentions n quantifiers, and consider the formula $Q\phi$, where Q is one of $(\forall x), (\exists x), (\forall a), (\exists a), (\forall s)$. Here, x, a, and s are individual variables of sort *object*, *action* and *situation*, respectively. When Q is $(\exists s)$, we reason as follows. Suppose that ν is an assignment to the variables of $\mathcal{L}_{sitcalc}$ of values from the domain of $\mathbb{S}_{pr}$ such that $\mathbb{S}_{pr}, \nu \models (\exists s)\phi$. Then for some assignment μ differing (if at all) from ν only in that it assigns an element from the *situation* domain of $\mathbb{S}_{pr}$ to s, $\mathbb{S}_{pr}, \mu \models \phi$. By induction hypothesis, $\mathbb{S}, \mu \models \phi$. Since the *situation* domain of $\mathbb{S}_{pr}$ is a subset of that for $\mathbb{S}$, it follows that $\mathbb{S}, \nu \models (\exists s)\phi$.

In the remaining four cases, namely when Q is one of $(\forall x), (\exists x), (\forall a), (\exists a)$, the proof is similar, but here we exploit the fact that $\mathbb{S}$ and $\mathbb{S}_{pr}$ share the same domains for sorts *object* and *action* in order to prove the cases $(\exists x)$ and $(\exists a)$. ■

[9] Recall that we use $\xi^{\mathbb{S}}$ for the denotation of the symbol ξ in a structure $\mathbb{S}$.

Definition 5.2 $\exists s$ Sentence.
A sentence of $\mathcal{L}_{sitcalc}$ is said to be an $\exists s$ sentence iff it has a prenex normal form of the form

$$\vec{Q}_1(\exists s_1)\vec{Q}_2(\exists s_2)\cdots(\exists s_n)\vec{Q}_n\psi, \quad n \geq 0,$$

where $s_1, \ldots, s_n$ are variables of sort *situation*, ψ is quantifier free, and each $\vec{Q}_i$ is a sequence of zero or more quantifiers over variables of sort $action \cup object$.

It will be for $\exists s$ sentences that we prove various Σ-reduction theorems, i.e. theorems that justify ignoring certain axioms of Σ in establishing that $\exists s$ sentences are entailed by a basic action theory.

Corollary 5.1 *If ϕ is a first order $\exists s$ sentence of $\mathcal{L}_{sitcalc}$ and $\mathbb{S}$ is a structure for $\mathcal{L}_{sitcalc}$, then whenever $\mathbb{S}_{pr}$ satisfies ϕ, so also does $\mathbb{S}$.*

Definition 5.3 Adjunct Structure.
Let $\mathbb{S}$ be a structure for $\mathcal{L}_{sitcalc}$. The *adjunct* structure of $\mathbb{S}$, $\mathbb{S}_{adj}$, is that structure for $\mathcal{L}_{sitcalc}$ defined as follows:

1. $\mathbb{S}_{adj}$ has the same domains of sort *action* and *object* as does $\mathbb{S}$ (and therefore, as does $\mathbb{S}_{pr}$).

2. $\mathbb{S}_{adj}$ has as its *situation* domain the set of all finite sequences of elements from its *action* domain, as in the proof of Theorem 1. Over this domain, the function symbols S_0 and do, and the predicate symbol $\sqsubset$ are interpreted as follows:

   $S_0^{\mathbb{S}_{adj}} = [\,]$

   $do^{\mathbb{S}_{adj}}(\alpha, [\alpha_1, \ldots, \alpha_n]) = [\alpha_1, \ldots, \alpha_n, \alpha]$.

   $\sigma \sqsubset^{\mathbb{S}_{adj}} \sigma'$ iff the sequence σ is a proper initial subsequence of σ'.

   With this domain of situations and interpretation for S_0, do and $\sqsubset$, $\mathbb{S}_{adj}$ is a model of Σ.

3. Define a function τ from the domain of $\mathbb{S}_{adj}$ onto that of $\mathbb{S}_{pr}$ as follows:

   For elements $[\alpha_1, \ldots, \alpha_n]$, $n \geq 0$, from the *situation* domain of $\mathbb{S}_{adj}$,

   $\tau([\,]) = S_0^{\mathbb{S}}$,

   and for $n \geq 1$,

   $\tau([\alpha_1, \ldots, \alpha_n]) = do^{\mathbb{S}}(\alpha_n, \tau([\alpha_1, \ldots, \alpha_{n-1}]))$.

   For elements d from the *action* and *object* domains of $\mathbb{S}_{adj}$ (and therefore of $\mathbb{S}$ and $\mathbb{S}_{pr}$),

   $\tau(d) = d$.

4. Next, we specify how $\mathbb{S}_{adj}$ interprets predicate symbols of $\mathcal{L}_{sitcalc}$ other than $\sqsubset$, and function symbols of $\mathcal{L}_{sitcalc}$ other than do and S_0.

(a) *Predicate symbols:* Suppose P is an n-ary predicate symbol of $\mathcal{L}_{sitcalc}$ other than $\sqsubset$. Then

$$(d_1, \ldots, d_n) \in P^{\mathbb{S}_{adj}} \text{ iff } (\tau(d_1), \ldots, \tau(d_n)) \in P^{\mathbb{S}}.$$

Here, $d_1, \ldots, d_n$ are elements from the domain of $\mathbb{S}_{adj}$ of the right sorts appropriate for P.

(b) *Function symbols:* Suppose f is an n-ary predicate symbol of $\mathcal{L}_{sitcalc}$ other than S_0 and do. Then

$$f^{\mathbb{S}_{adj}}(d_1, \ldots, d_n) = f^{\mathbb{S}}(\tau(d_1), \ldots, \tau(d_n)).$$

As before, $d_1, \ldots, d_n$ are elements from the domain of $\mathbb{S}_{adj}$ of the right sorts appropriate for f. Because all functions of $\mathcal{L}_{sitcalc}$ other than S_0 and do takes values of sort $action \cup object$, and because $\mathbb{S}_{adj}$ and $\mathbb{S}$ have the same *action* and *object* domains, this specification is well defined.

Remark 2 *τ is a homomorphism (Enderton [5]) from $\mathbb{S}_{adj}$ onto $\mathbb{S}_{pr}$ for the reduced language $\mathcal{L}_{sitcalc}$ without the predicate symbol $\sqsubset$. In other words, τ preserves all relations between $\mathbb{S}_{adj}$ and $\mathbb{S}_{pr}$, except for $\sqsubset$, and it preserves all functions.*

Corollary 5.2 *Suppose that $\mathbb{S}$ is a structure for $\mathcal{L}_{sitcalc}$. Suppose further that ϕ is a first order formula of $\mathcal{L}_{sitcalc}$ that does not mention the predicate symbol $\sqsubset$, and that does not mention an equality atom over terms of sort* situation.[10] *Then,*

$$\mathbb{S}_{adj}, \nu \models \phi \text{ iff } \mathbb{S}_{pr}, \tau(\nu) \models \phi.$$

Here, $\tau(\nu)$ is that variable assignment assigning $\tau(d)$ to a variable whenever ν assigns d to that variable, where d is an element of the domain of $\mathbb{S}_{adj}$.

Proof: By the homomorphism theorem for first order logic (Enderton [5]). The result holds even when ϕ is permitted to mention equality atoms over terms of sort *action* or *object* because $\mathbb{S}_{adj}$ and $\mathbb{S}_{pr}$ share the same domains of sorts *action* and *object*. ∎

Definition 5.4 By $\Sigma_=$ we mean the foundational axiom (1) together with the following sentence:

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

It is easy to verify that this sentence is a logical consequence of $\Sigma - \{Induction\}$, by which we mean Σ without the induction axiom (2), so there is nothing new here insofar as the foundational axioms are concerned. However, for the purposes of isolating the equality foundations for situations, it is necessary to make it explicit.

Lemma 5.2 *If $\mathbb{M}$ is a model of $\Sigma_=$, then for all finite sequences σ, σ' of elements from the* action *domain of $\mathbb{M}$,*

[10] ϕ is permitted to mention equality atoms over terms of sort *action* or *object*.

$\tau(\sigma) = \tau(\sigma')$ *iff* $\sigma = \sigma'$.[11]

Proof: The $\Leftarrow$ direction is immediate. For the other direction, suppose $\sigma = [\alpha_1, \ldots, \alpha_m]$ and $\sigma' = [\beta_1, \ldots, \beta_n]$. We proceed by induction on m.

Suppose $m = 0$. If $n > 0$, then

$$\tau([\,]) = S_0^{\mathbb{M}} = \tau([\beta_1, \ldots, \beta_n]) = do^{\mathbb{M}}(\beta_n, \tau([\beta_1, \ldots, \beta_{n-1}]),$$

and this is impossible because $\mathbb{M}$ is a model of $\Sigma_=$. Therefore, $n = 0$ and this establishes the base case for the inductive proof.

For the inductive step, assume the result for all $m > 0$, and suppose that

$$\tau([\alpha_1, \ldots, \alpha_{m+1}]) = \tau([\beta_1, \ldots, \beta_n]).$$

The case $n = 0$ is impossible by the same argument as before. Therefore, $n > 0$, and we have

$$\tau([\alpha_1, \ldots, \alpha_{m+1}]) = do^{\mathbb{M}}(\alpha_{m+1}, \tau([\alpha_1, \ldots, \alpha_m]) = \tau([\beta_1, \ldots, \beta_n]) = do^{\mathbb{M}}(\beta_n, \tau([\beta_1, \ldots, \beta_{n-1}]),$$

Since $\mathbb{M}$ is a model of $\Sigma_=$, we have

$$\alpha_{m+1} = \beta_n \text{ and } \tau([\alpha_1, \ldots, \alpha_m]) = \tau([\beta_1, \ldots, \beta_{n-1}]).$$

By the induction hypothesis, $[\alpha_1, \ldots, \alpha_m] = [\beta_1, \ldots, \beta_{n-1}]$, and therefore, $[\alpha_1, \ldots, \alpha_{m+1}] = [\beta_1, \ldots, \beta_n]$. ■

Corollary 5.3 *If $\mathbb{M}$ is a model of $\Sigma_=$, then τ is an isomorphism between $\mathbb{M}_{adj}$ and $\mathbb{M}_{pr}$ for the reduced language $\mathcal{L}_{sitcalc}$ without the predicate symbol $\sqsubset$.*

Lemma 5.3 *If $\mathbb{M}$ is a model of $\Sigma - \{Induction\}$, then for all finite sequences σ, σ' of elements from the* action *domain of $\mathbb{M}$,*

$$\sigma \sqsubset^{\mathbb{M}_{adj}} \sigma' \textit{ iff } \tau(\sigma) \sqsubset^{\mathbb{M}} \tau(\sigma').$$

Proof: We must prove that for all finite sequences σ, σ' of elements from the *action* domain of $\mathbb{M}$,

$$\tau(\sigma) \sqsubset^{\mathbb{M}} \tau(\sigma') \text{ iff } \sigma \text{ is a proper subsequence of } \sigma'.$$

($\Rightarrow$) Suppose, for $m, n \geq 0$, that $\sigma = [\alpha_1, \ldots, \alpha_m]$ and $\sigma' = [\beta_1, \ldots, \beta_n]$. The proof is by induction on n. If $n = 0$, then $\tau(\sigma) \sqsubset^{\mathbb{M}} S_0^{\mathbb{M}}$, which is impossible since $\mathbb{M}$ satisfies foundational axiom (3). Therefore, the base case for the induction is $n = 1$. In this case, suppose that $m \geq 1$. Then $do^{\mathbb{M}}(\alpha_m, \tau([\alpha_1, \ldots, \alpha_{m-1}])) \sqsubset^{\mathbb{M}} do^{\mathbb{M}}(\beta_1, S_0^{\mathbb{M}})$. Since $\mathbb{M}$ satisfies axiom (4), we conclude that $do^{\mathbb{M}}(\alpha_m, \tau([\alpha_1, \ldots, \alpha_{m-1}])) \sqsubseteq^{\mathbb{M}} S_0^{\mathbb{M}}$, and this is impossible because $\mathbb{M}$ satisfies axiom (3) and the sentence $S_0 \neq do(a, s)$. Therefore, $m = 0$ and the base case for the induction is established.

For the inductive step, assume the result for $n \geq 1$, and suppose that $\tau(\sigma) \sqsubset^{\mathbb{M}} \tau([\beta_1, \ldots, \beta_{n+1}])$. Therefore, $\tau(\sigma) \sqsubset^{\mathbb{M}} do^{\mathbb{M}}(\beta_{n+1}, \tau([\beta_1, \ldots, \beta_n])$. Since $\mathbb{M}$ satisfies axiom (4), we conclude that $\tau(\sigma) \sqsubseteq^{\mathbb{M}} \tau([\beta_1, \ldots, \beta_n])$.

[11] Recall that τ is the mapping from the domain of $\mathbb{M}_{adj}$ onto that of $\mathbb{M}_{pr}$, as described in Definition 5.3.

Case 1: $\tau(\sigma) = \tau([\beta_1, \ldots, \beta_n])$. Since $\mathbb{M}$ is a model of $\mathcal{D} - \{Induction\}$, it is a model of $\Sigma_=$. Therefore, by Lemma 5.2, $\sigma = [\beta_1, \ldots, \beta_n])$, and therefore, σ is a proper subsequence of $[\beta_1, \ldots, \beta_{n+1}]$.

Case 2: $\tau(\sigma) \sqsubset^{\mathbb{M}} \tau([\beta_1, \ldots, \beta_n])$. By the induction hypothesis, σ is a proper subsequence of $[\beta_1, \ldots, \beta_n]$, and therefore is a proper subsequence of $[\beta_1, \ldots, \beta_{n+1}]$.

($\Leftarrow$) Suppose for $m \geq 0$ and $n \geq 1$ that $\sigma = [\alpha_1, \ldots, \alpha_m]$ and $\sigma' = [\alpha_1, \ldots, \alpha_m, \beta_1, \ldots, \beta_n]$. We prove that $\tau(\sigma) \sqsubset^{\mathbb{M}} \tau(\sigma')$. The proof is by induction on n. When $n = 1$, $\tau(\sigma') = do^{\mathbb{M}}(\beta_1, \tau(\sigma))$, and because $\mathbb{M}$ is a model of $\Sigma - \{Induction\}$, it satisfies the sentence $(\forall a, s) s \sqsubset do(a, s)$. Therefore, $\tau(\sigma) \sqsubset^{\mathbb{M}} \tau(\sigma')$.

For the inductive step, assume the result for $n \geq 1$, and suppose that $\sigma' = [\alpha_1, \ldots, \alpha_m, \beta_1, \ldots, \beta_{n+1}]$. Then $\tau(\sigma') = do^{\mathbb{M}}(\beta_{n+1}, \tau([\alpha_1, \ldots, \alpha_m, \beta_1, \ldots, \beta_n])$. By induction, $\tau(\sigma) \sqsubset^{\mathbb{M}} \tau([\alpha_1, \ldots, \alpha_m, \beta_1, \ldots, \beta_n])$. Since $\mathbb{M}$ satisfies axiom (4), $\tau(\sigma) \sqsubset^{\mathbb{M}} \tau(\sigma')$. ■

Lemma 5.4 *Let $\mathcal{D} = \Sigma \cup \mathcal{D}_{ss} \cup \mathcal{D}_{ap} \cup \mathcal{D}_{una} \cup \mathcal{D}_{S_0}$ be a basic action theory. Whenever $\mathbb{M}$ is a model of $\mathcal{D}_{ss} \cup \mathcal{D}_{ap} \cup \mathcal{D}_{una} \cup \mathcal{D}_{S_0}$, $\mathbb{M}_{adj}$ is a model of $\mathcal{D}$.*

Proof: By its construction, $\mathbb{M}_{adj}$ is a model of Σ. All sentences of $\mathcal{D}_{ss} \cup \mathcal{D}_{ap} \cup \mathcal{D}_{una} \cup \mathcal{D}_{S_0}$ are universally quantified over situations, and therefore, are logically equivalent to the negations of first order $\exists s$ sentences. Since $\mathbb{M}$ satisfies each of these sentences, then by Corollary 5.1, so does $\mathbb{M}_{pr}$. By the definitions of Section 2.3 specifying the syntactic forms that these sentences must possess, none of these sentences mentions $\sqsubset$ or an equality atom over terms of sort *situation*. The result now follows from Corollary 5.2. ■

Theorem 4 (Σ-Reduction Theorem)
Suppose that $\mathcal{D} = \Sigma \cup \mathcal{D}_{ss} \cup \mathcal{D}_{ap} \cup \mathcal{D}_{una} \cup \mathcal{D}_{S_0}$ is a basic action theory and that ϕ is a first order $\exists s$ sentence.

1. $\mathcal{D} \models \phi$ *iff* $\mathcal{D} - \{Induction\} \models \phi$.

2. *If ϕ does not mention the predicate symbol $\sqsubset$,*

   $\mathcal{D} \models \phi$ *iff* $\Sigma_= \cup \mathcal{D}_{ss} \cup \mathcal{D}_{ap} \cup \mathcal{D}_{una} \cup \mathcal{D}_{S_0} \models \phi$.

3. *If ϕ does not mention the predicate symbol $\sqsubset$ and it does not mention an equality atom over terms of sort* situation,

   $\mathcal{D} \models \phi$ *iff* $\mathcal{D}_{ss} \cup \mathcal{D}_{ap} \cup \mathcal{D}_{una} \cup \mathcal{D}_{S_0} \models \phi$.

Proof:

1. Suppose $\mathcal{D} \models \phi$, but $\mathcal{D} - \{Induction\} \not\models \phi$. Then there is a model $\mathbb{M}$ of $\mathcal{D} - \{Induction\}$ that satisfies $\neg\phi$. By Corollary 5.1 $\mathbb{M}_{pr}$ satisfies $\neg\phi$. By Corollary 5.3 and Lemma 5.3, $\mathbb{M}_{adj}$ is isomorphic to $\mathbb{M}_{pr}$, and therefore $\mathbb{M}_{adj}$ satisfies $\neg\phi$. By Lemma 5.4, $\mathbb{M}_{adj}$ is a model of $\mathcal{D}$, contradiction.

2. Suppose $\mathcal{D} \models \phi$, but $\Sigma_= \cup \mathcal{D}_{ss} \cup \mathcal{D}_{ap} \cup \mathcal{D}_{una} \cup \mathcal{D}_{S_0} \not\models \phi$. Then there is a model $\mathbb{M}$ of $\Sigma_= \cup \mathcal{D}_{ss} \cup \mathcal{D}_{ap} \cup \mathcal{D}_{una} \cup \mathcal{D}_{S_0}$ that satisfies $\neg\phi$. By Corollary 5.1 $\mathbb{M}_{pr}$ satisfies $\neg\phi$. By Corollary 5.3, $\mathbb{M}_{adj}$ satisfies $\neg\phi$. By Lemma 5.4, $\mathbb{M}_{adj}$ is a model of $\mathcal{D}$, contradiction.

3. Suppose $\mathcal{D} \models \phi$, but $\mathcal{D}_{ss} \cup \mathcal{D}_{ap} \cup \mathcal{D}_{una} \cup \mathcal{D}_{S_0} \not\models \phi$. Then there is a model $\mathbb{M}$ of $\mathcal{D}_{ss} \cup \mathcal{D}_{ap} \cup \mathcal{D}_{una} \cup \mathcal{D}_{S_0}$ that satisfies $\neg\phi$. By Corollary 5.1 $\mathbb{M}_{pr}$ satisfies $\neg\phi$. By Corollary 5.2, $\mathbb{M}_{adj}$ satisfies $\neg\phi$. By Lemma 5.4, $\mathbb{M}_{adj}$ is a model of $\mathcal{D}$, contradiction.

■

6 Σ-Elimination for GOLOG

GOLOG [18] is a situation calculus-based logic programming language for implementing simulators and controllers for dynamical systems using a repertoire of user specified primitive actions. It has been used for agent programming [16], robotics [3, 35, 14], and computer animation [6]. GOLOG provides the usual kinds of imperative programming language control structures as well as two flavors of nondeterministic choice:

1. *Sequence:* $\alpha\,;\,\beta$. Do program α, followed by program β.
2. *Test actions:* ϕ? Test the truth value of expression ϕ in the current situation.
3. *Nondeterministic choice:* $\alpha \mid \beta$. Do program α or program β.
4. *Nondeterministic choice of arguments:* $(\pi\ x)\alpha$. Nondeterministically pick a value for x, and for that value of x, do the program α.
5. *Procedures*, including recursion.

As will be described below, a GOLOG program is evaluated by a theorem-prover relative to a basic theory of actions describing the dynamics of the application domain. These axioms include the foundational axioms for the situation calculus. The purpose of this section is to prove that these foundational axioms are not required for evaluating GOLOG programs.

6.1 The Syntax of GOLOG Programs

For the purposes of specifying the syntax of the programming language GOLOG, we require a language that is just like $\mathcal{L}_{sitcalc}$, except it suppresses all references to situations.

Definition 6.1 Situation-Suppressed Terms and Formulas
The *situation-suppressed* terms obtained from $\mathcal{L}_{sitcalc}$ are inductively defined by:

1. Any variable of $\mathcal{L}_{sitcalc}$ of sort *action* or *object* is a situation-suppressed term.

2. If f is an $(n+1)$-ary functional fluent of $\mathcal{L}_{sitcalc}$, and $t_1, \ldots, t_n$ are situation-suppressed terms of sorts appropriate for the first n arguments of f, then $f(t_1, \ldots, t_n)$ is a situation-suppressed term.

3. If g is an m-ary non-functional fluent symbol of $\mathcal{L}_{sitcalc}$ other than S_0 or do, and $t_1, \ldots, t_m$ are *situation-suppressed* terms of sorts appropriate for the arguments of g, then $g(t_1, \ldots, t_m)$ is a situation-suppressed term.

The *situation-suppressed* formulas obtained from $\mathcal{L}_{sitcalc}$ are inductively defined by:

1. Whenever t and t' are situation-suppressed terms of the same sort, then $t = t'$ is a situation-suppressed formula. Because situation-suppressed terms are never of sort *situation*, the situation-suppressed formulas never mention an equality atom between terms of sort *situation*.

2. When t is a situation-suppressed term of sort *action*, then $Poss(t)$ is a situation-suppressed formula.

3. When F is an $(n+1)$-ary relational fluent symbol of $\mathcal{L}_{sitcalc}$ and $t_1, \ldots, t_n$ are situation-suppressed terms of sorts appropriate for the first n arguments of F, then $F(t_1, \ldots, t_n)$ is a situation-suppressed formula.

4. When P is an m-ary non-fluent predicate symbol of $\mathcal{L}_{sitcalc}$ other than $\sqsubset$, and $t_1, \ldots, t_m$ are situation-suppressed terms of sorts appropriate for the arguments of P, then $P(t_1, \ldots, t_m)$ is a situation-suppressed formula.

5. When ϕ and ψ are situation-suppressed formulas, so are $\neg\phi$ and $\phi \wedge \psi$. When v is a variable of $\mathcal{L}_{sitcalc}$ of sort *action* or *object*, then $(\exists v)\phi$ is a situation-suppressed formula.

Therefore, situation-suppressed formulas are first order, never quantify over situations, never mention $\sqsubset$, nor do they ever mention terms of sort *situation*.

We can now describe the syntax of GOLOG programs:

$$
\begin{array}{lll}
\langle program \rangle & ::= & \langle primitive\ \ action \rangle \mid \\
& & \langle test\ \ condition \rangle ? \mid \\
& & (\langle program \rangle ; \langle program \rangle) \mid \\
& & (\langle program \rangle \mid \langle program \rangle) \mid \\
& & (\pi x)\langle program \rangle \mid \\
& & \langle procedure\ \ call \rangle \mid \\
& & (\textbf{proc}\ P_1\,(\vec{v}_1)\ \langle program \rangle\ \textbf{endProc}\ ; \\
& & \qquad\qquad \vdots \\
& & \ \ \textbf{proc}\ P_n\,(\vec{v}_n)\ \langle program \rangle\ \textbf{endProc}\ ; \\
& & \ \ \langle program \rangle)
\end{array}
$$

Here,

1. $\langle primitive\ action \rangle$ is a situation-suppressed term of sort *action*.

2. $\langle test\ condition \rangle$ is a situation-suppressed formula.

3. In $(\pi x)\langle program\rangle$, x must be a variable of $\mathcal{L}_{sitcalc}$ of sort *action* or *object*.

4. $\langle procedure\ call\rangle$ must have the syntactic form $P(t_1, \ldots, t_n)$, where P is an $(n+2)$-ary predicate variable of $\mathcal{L}_{sitcalc}$ whose first n arguments are not of sort *situation*, and whose last two arguments are of sort *situation*. Moreover, $t_1, \ldots, t_n$ must be situation-suppressed terms whose sorts are appropriate for the first n arguments of P.

5. In a procedure declaration **proc** $P\,(\vec{v})\ \langle program\rangle$ **endProc**, P must be an $(n+2)$-ary predicate variable of $\mathcal{L}_{sitcalc}$ whose first n arguments are not of sort *situation*, and whose last two arguments are of sort *situation*. Moreover, $\vec{v}$ must be variables of $\mathcal{L}_{sitcalc}$ whose sorts are appropriate for the first n arguments of P.

Notice that GOLOG provides for arbitrary nesting of blocks with procedure declarations local to their block. Other control structures, e.g. conditionals and while loops, can be defined in terms of the above constructs:

$$\textbf{if}\ \langle test\ \ condition\rangle\ \textbf{then}\ \langle program1\rangle\ \textbf{else}\ \langle program2\rangle \stackrel{def}{=} \langle test\ \ condition\rangle? \ ; \ \langle program1\rangle \mid \neg\langle test\ \ condition\rangle? \ ; \ \langle program2\rangle$$

To define **while** loops, first introduce a nondeterministic iteration operator *, where $\langle program\rangle^*$ means do $\langle program\rangle$ 0 or more times:

$$\langle program\rangle^* \stackrel{def}{=} \textbf{proc}\ P()\ true? \mid [\langle program\rangle \, ; P()]\ \textbf{endProc}\, ; P()$$

Then **while** loops can be defined in terms of the * operator:

$$\textbf{while}\ \langle test\ \ condition\rangle\ \textbf{do}\ \langle program\rangle\ \textbf{endWhile} \stackrel{def}{=} [\langle test\ \ condition\rangle? \ ; \ \langle program\rangle]^* ; \neg\langle test\ \ condition\rangle?$$

Just as conventional Algol-like programming languages never explicitly refer to the state of their computation, GOLOG programs never explicitly mention situations, hence the emphasis above on situation-suppressed terms and formulas. In defining the semantics of such programs, these situations are taken into account, as we now describe.

6.2 The Semantics of GOLOG

Definition 6.2 Restoring Suppressed Situation Arguments
Whenever t is a situation-suppressed term and σ is a term of $\mathcal{L}_{sitcalc}$ of sort *situation*, $t[\sigma]$ denotes that term of $\mathcal{L}_{sitcalc}$ obtained by restoring the term σ as the *situation* argument to all of the functional fluents terms mentioned by t. In addition, whenever ϕ is a situation-suppressed formula, $\phi[\sigma]$ denotes that formula of $\mathcal{L}_{sitcalc}$ obtained by restoring the term σ as the *situation* argument to all of the functional fluent terms and all of the relational fluent atoms mentioned by ϕ.

Next, we define an abbreviation $Do(\delta, \sigma, \sigma')$, where δ is a program expression and σ and σ' are terms of sort *situation*. $Do(\delta, \sigma, \sigma')$ should be viewed as a macro that expands into a second order situation calculus formula; moreover, *that formula says that situation σ' can* *be reached from situation σ by executing some sequence of actions specified by δ.* Note that our programs may be nondeterministic, that is, may have multiple executions terminating in different situations.

Do is defined inductively on the structure of its first argument as follows:

1. *Primitive actions:* When α is a situation-suppressed term of sort *action*,

$$Do(\alpha, \sigma, \sigma') \stackrel{def}{=} Poss(\alpha[\sigma], \sigma) \wedge \sigma' = do(\alpha[\sigma], \sigma).$$

2. *Test actions:* When ϕ is a situation-suppressed formula,

$$Do(\phi?, \sigma, \sigma') \stackrel{def}{=} \phi[\sigma] \wedge \sigma = \sigma'.$$

3. *Sequence:*

$$Do(\delta_1; \delta_2, \sigma, \sigma') \stackrel{def}{=} (\exists s).Do(\delta_1, \sigma, s) \wedge Do(\delta_2, s, \sigma').$$

4. *Nondeterministic choice of two actions:*

$$Do(\delta_1 \mid \delta_2, \sigma, \sigma') \stackrel{def}{=} Do(\delta_1, \sigma, \sigma') \vee Do(\delta_2, \sigma, \sigma').$$

5. *Nondeterministic choice of action arguments:*

$$Do((\pi x)\, \delta, \sigma, \sigma') \stackrel{def}{=} (\exists x)\, Do(\delta, \sigma, \sigma').$$

6. *Procedure calls:* For any predicate variable P of arity $n+2$ whose first n arguments are not of sort *situation*, and whose last two arguments are of sort *situation*:

$$Do(P(t_1, \ldots, t_n), \sigma, \sigma') \stackrel{def}{=} P(t_1[\sigma], \ldots, t_n[\sigma], \sigma, \sigma').$$

7. *Blocks with local procedure declarations:*

$$\begin{aligned} &Do(\{\mathbf{proc}\ P_1\ (\vec{v}_1)\ \delta_1\ \mathbf{endProc}\ ;\ \cdots\ ;\ \mathbf{proc}\ P_n\ (\vec{v}_n)\ \delta_n\ \mathbf{endProc}\ ;\ \delta_0\}, \sigma, \sigma') \\ &\quad \stackrel{def}{=} (\forall P_1, \ldots, P_n).[\bigwedge_{i=1}^{n} (\forall s, s', \vec{v}_i).Do(\delta_i, s, s') \supset P_i(\vec{v}_i, s, s')] \\ &\qquad\qquad \supset Do(\delta_0, \sigma, \sigma'). \end{aligned}$$

Notice that in the definition for procedure calls, the actual parameters (t_i) are first evaluated with respect to the current situation σ ($t_i[\sigma]$) before passing them to the procedure P, so GOLOG's procedure invocation is *call by value*.

Except for procedures, the above macro approach to defining the semantics of GOLOG draws considerably from dynamic logic [10]. In effect, it reifies as situations in the object language of the situation calculus, the possible worlds with which the semantics of dynamic logic is defined. The macro definition for GOLOG procedures corresponds to the more usual Scott-Strachey *least fixed-point* definition in standard programming language semantics [39].

With the above definition of $Do(\delta, \sigma, \sigma')$ in hand, we are in a position to define what we mean by the evaluation of a GOLOG program.

6.3 The Evaluation of GOLOG Programs

Definition 6.3 Proper GOLOG Program
A GOLOG program δ is proper iff s and s' are the only free variables (individual or predicate) mentioned in $Do(\delta, s, s')$. In other words, for each individual variable x mentioned in δ, x lies in the scope of $(\forall x)$ or $(\exists x)$ in a test condition of δ, or x is mentioned in the scope of a nondeterministic choice operator $(\pi\, x)$. Moreover, for every procedure call $P(t_1, \ldots, t_n)$ mentioned in δ, the second order variable P is bound in δ by a procedure declaration for P in some block surrounding the procedure call.

A proper GOLOG program is evaluated relative to a background basic theory of actions specifying a particular application domain. Specifically, if $\mathcal{D}$ is such a basic action theory, and δ is a proper GOLOG program, then the evaluation of δ relative to $\mathcal{D}$ is defined to be the task of establishing the following entailment:

$$\mathcal{D} \models (\exists s) Do(\delta, S_0, s).$$

Any binding for the existentially quantified variable s obtained as a side effect of such a proof constitutes an execution trace of δ. In this respect, the evaluation of a GOLOG program is much like that of a Prolog goal statement, in the sense that both are evaluated by a theorem-prover for the purposes of obtaining bindings to existentially quantified variables of the theorem. Of course, in the case of GOLOG, this theorem-proving task is much more daunting, because in general, $(\exists s) Do(\delta, S_0, s)$ stands for a very complicated second order sentence of $\mathcal{L}_{sitcalc}$, and moreover, $\mathcal{D}$ includes a second order induction axiom. The purpose of the next section is to somewhat simplify this theorem-proving task for GOLOG, by showing that the foundational axioms of $\mathcal{D}$ are unnecessary.[12]

6.4 Σ-Elimination

Lemma 6.1 *Suppose that $\mathbb{S}$ is a structure for $\mathcal{L}_{sitcalc}$. Then for every GOLOG program δ, possibly with free individual and predicate variables, and for every assignment ν to the individual and predicate variables of $\mathcal{L}_{sitcalc}$ of values from the domain of $\mathbb{S}_{pr}$,*

$$\mathbb{S}_{pr}, \nu \models Do(\delta, s, s') \quad \Rightarrow \quad \mathbb{S}, \nu \models Do(\delta, s, s').$$

Proof: By induction on the syntactic form of δ.

1. Suppose δ is a primitive action term α. Then

   $$Do(\delta, s, s') \stackrel{def}{=} Poss(\alpha[s], s) \wedge s' = do(\alpha[s], s).$$

   This case is covered by Lemma 5.1.

[12] In [18, 30], an implementation for a GOLOG interpreter is given, written in Prolog, but because it relies on Prolog's negation-as-failure mechanism, this implementation is suitable only when the background basic action theory provides complete information about its initial database. When this is not the case, in robotics for example, more general theorem-proving techniques than Prolog's are required, and it is towards simplifying the theorem-proving task for such general GOLOG interpreters that the results of this section are directed.

2. Suppose δ is a test action $\phi?$. Then

   $$Do(\delta, s, s') \stackrel{def}{=} \phi[s] \wedge s' = s.$$

   Since the formula $\phi[s]$ does not quantify over situations, this case is also covered by Lemma 5.1.

3. Suppose δ is $\delta_1; \delta_2$. Then

   $$Do(\delta, s, s') \stackrel{def}{=} (\exists s'').Do(\delta_1, s, s'') \wedge Do(\delta_2, s'', s').$$

   Suppose $\mathbb{S}_{pr}, \nu \models (\exists s'').Do(\delta_1, s, s'') \wedge Do(\delta_2, s'', s')$, where ν assigns elements from the domain of $\mathbb{S}_{pr}$ to the variables of $\mathcal{L}_{sitcalc}$. Then for some μ differing (if at all) from ν only in that it assigns an element from the *situation* domain of $\mathbb{S}_{pr}$ to s'', $\mathbb{S}_{pr}, \mu \models Do(\delta_1, s, s'') \wedge Do(\delta_2, s'', s')$. By induction hypothesis, $\mathbb{S}, \mu \models Do(\delta_1, s, s'') \wedge Do(\delta_2, s'', s')$. Since the *situation* domain of $\mathbb{S}_{pr}$ is a subset of that for $\mathbb{S}$, it follows that $\mathbb{S}, \nu \models (\exists s).Do(\delta_1, s, s'') \wedge Do(\delta_2, s'', s')$.

4. Suppose δ is $\delta_1 \mid \delta_2$. Then

   $$Do(\delta, s, s') \stackrel{def}{=} Do(\delta_1, s, s') \vee Do(\delta_2, s, s').$$

   The proof is immediate by induction.

5. Suppose δ is $(\pi\ x)\gamma$. Then x must be a variable of sort *action* or *object* and

   $$Do(\delta, s, s') \stackrel{def}{=} (\exists x)Do(\gamma, s, s').$$

   The proof is immediate by induction, and the fact that the *action* and *object* domains of $\mathbb{S}$ and $\mathbb{S}_{pr}$ are the same.

6. Suppose δ is the procedure call $P(t_1, \ldots, t_n)$, where P is a predicate variable of arity $n+2$ whose first n arguments are not of sort *situation*, and whose last two arguments are of sort *situation*. Then,

   $$Do(\delta, s, s') \stackrel{def}{=} P(t_1[s], \ldots, t_n[s], s, s').$$

   The result follows because ν assigns elements from the domain of $\mathbb{S}_{pr}$ to the individual and predicate variables of $\mathcal{L}_{sitcalc}$, and $\mathbb{S}_{pr}$ has the same *action* and *object* domains as $\mathbb{S}$, and the *situation* domain of the former is a subset of that of the latter.

7. Suppose δ is

   $$\textbf{proc}\ P_1\ (\vec{v}_1)\ \delta_1\ \textbf{endProc}\ ;\ \cdots\ ;\ \textbf{proc}\ P_n\ (\vec{v}_n)\ \delta_n\ \textbf{endProc}\ ;\ \delta_0$$

   Assume, where ν assigns elements from the domain of $\mathbb{S}_{pr}$ to the individual variables of $\mathcal{L}_{sitcalc}$, that $\mathbb{S}_{pr}, \nu \models Do(\delta, s, s')$. Then,

   $$\begin{aligned} \mathbb{S}_{pr}, \nu \models (\forall P_1, \ldots, P_n).[\bigwedge_{i=1}^{n} (\forall s_1, s_2, \vec{v}_i).Do(\delta_i, s_1, s_2) \supset P_i(\vec{v}_i, s_1, s_2)] \\ \supset Do(\delta_0, s, s'). \end{aligned} \tag{20}$$

For $i = 1, \ldots, n$, let the predicate variable P_i have arity $k_i + 2$, and let $\mathbf{P}_i \subseteq (action \cup object)^{k_i} \times Sit \times Sit$ be any fixed relation over the domain of $\mathbb{S}$, where Sit is the *situation* domain of $\mathbb{S}$. We must prove that

$$\mathbb{S}, \nu \models [\bigwedge_{i=1}^{n} (\forall s_1, s_2, \vec{v}_i).Do(\delta_i, s_1, s_2) \supset \mathbf{P}_i(\vec{v}_i, s_1, s_2)] \supset Do(\delta_0, s, s').$$

So, for $i = 1, \ldots, n$, assume that

$$\mathbb{S}, \nu \models (\forall s_1, s_2, \vec{v}_i).Do(\delta_i, s_1, s_2) \supset \mathbf{P}_i(\vec{v}_i, s_1, s_2). \tag{21}$$

We must prove that

$$\mathbb{S}, \nu \models Do(\delta_0, s, s'). \tag{22}$$

Let $\mathbf{R}_i = \mathbf{P}_i \cap (action \cup object)^{k_i} \times Sit_{pr} \times Sit_{pr}$, where Sit_{pr} is the situation domain of $\mathbb{S}_{pr}$. Since $\mathbb{S}_{pr}$ and $\mathbb{S}$ share the same domains of sort *action* and *object*, and since the *situation* domain of $\mathbb{S}_{pr}$ is a subset of that of $\mathbb{S}$, $\mathbf{R}_i$ is a relation over the domain of $\mathbb{S}_{pr}$. We prove that, for $i = 1, \ldots, n$

$$\mathbb{S}_{pr}, \nu \models (\forall s_1, s_2, \vec{v}_i).Do(\delta_i, s_1, s_2) \supset \mathbf{R}_i(\vec{v}_i, s_1, s_2). \tag{23}$$

From this, and (20), it will follow that $\mathbb{S}_{pr}, \nu \models Do(\delta_0, s, s')$, from which, by induction hypothesis, (22) will follow.

Proof of (23): Let μ be a variable assignment that is just like ν except that it assigns elements from the domain of $\mathbb{S}_{pr}$ to the individual variables $\vec{v}_i, s_1, s_2$. Then we must prove that for any such μ,

$$\mathbb{S}_{pr}, \mu \models Do(\delta_i, s_1, s_2) \supset \mathbf{R}_i(\vec{v}_i, s_1, s_2).$$

Assume that $\mathbb{S}_{pr}, \mu \models Do(\delta_i, s_1, s_2)$. We prove that $\mathbb{S}_{pr}, \mu \models \mathbf{R}_i(\vec{v}_i, s_1, s_2)$. By induction hypothesis, $\mathbb{S}, \mu \models Do(\delta_i, s_1, s_2)$. Therefore, because the domain of $\mathbb{S}_{pr}$ is a subset of the domain of $\mathbb{S}$, and by (21), $\mathbb{S}, \mu \models \mathbf{P}_i(\vec{v}_i, s_1, s_2)$. By the definition of $\mathbf{R}_i$, $\mathbb{S}, \mu \models \mathbf{R}_i(\vec{v}_i, s_1, s_2)$. But μ assigns elements from the domain of $\mathbb{S}_{pr}$ to $\vec{v}_i, s_1, s_2$, so $\mathbb{S}_{pr}, \mu \models \mathbf{R}_i(\vec{v}_i, s_1, s_2)$.

■

Corollary 6.1 *Suppose that $\mathbb{S}$ is a structure for $\mathcal{L}_{sitcalc}$. Then for every proper GOLOG program δ,*

$$\mathbb{S}_{pr} \models (\exists s)Do(\delta, S_0, s) \quad \Rightarrow \quad \mathbb{S} \models (\exists s)Do(\delta, S_0, s).$$

Lemma 6.2 *Suppose that $\mathbb{S}$ is a structure for $\mathcal{L}_{sitcalc}$. Then for every GOLOG program δ, possibly with free individual and predicate variables, and for every assignment ν to the individual and predicate variables of $\mathcal{L}_{sitcalc}$ of values from the domain of $\mathbb{S}_{adj}$,*

$\mathbb{S}_{adj}, \nu \models Do(\delta, s, s') \;\;\Rightarrow\;\; \mathbb{S}_{pr}, \tau(\nu) \models Do(\delta, s, s').$ [13]

Proof: By induction on the syntactic form of δ.

1. Suppose δ is a primitive action term α. Then

   $$Do(\delta, s, s') \stackrel{def}{=} Poss(\alpha[s], s) \wedge s' = do(\alpha[s], s).$$

   Suppose $\mathbb{S}_{adj}, \nu \models Poss(\alpha[s], s) \wedge s' = do(\alpha[s], s)$. By Corollary 5.2, $\mathbb{S}_{pr}, \tau(\nu) \models Poss(\alpha[s], s)$, and trivially, $\mathbb{S}_{pr}, \tau(\nu) \models s' = do(\alpha[s], s)$, so we are done.

2. Suppose δ is a test action $\phi?$. Then

   $$Do(\delta, s, s') \stackrel{def}{=} \phi[s] \wedge s' = s.$$

   Suppose $\mathbb{S}_{adj}, \nu \models \phi[s] \wedge s' = s$. The formula $\phi[s]$ does not mention the predicate symbol $\sqsubset$; neither does it mention an equality atom on situations. Therefore, by Corollary 5.2, $\mathbb{S}_{pr}, \tau(\nu) \models \phi[s]$, and trivially, $\mathbb{S}_{pr}, \tau(\nu) \models s' = s$, and we are done.

3. Suppose δ is $\delta_1; \delta_2$. Then

   $$Do(\delta, s, s') \stackrel{def}{=} (\exists s'').Do(\delta_1, s, s'') \wedge Do(\delta_2, s'', s').$$

   Suppose $\mathbb{S}_{adj}, \nu \models (\exists s'').Do(\delta_1, s, s'') \wedge Do(\delta_2, s'', s')$, where ν assigns elements from the domain of $\mathbb{S}_{adj}$ to the variables of $\mathcal{L}_{sitcalc}$. Then for some μ differing (if at all) from ν only in that it assigns an element from the *situation* domain of $\mathbb{S}_{adj}$ to s'', $\mathbb{S}_{adj}, \mu \models Do(\delta_1, s, s'') \wedge Do(\delta_2, s'', s')$. By induction hypothesis, $\mathbb{S}_{pr}, \tau(\mu) \models Do(\delta_1, s, s'') \wedge Do(\delta_2, s'', s')$. Therefore, $\mathbb{S}_{pr}, \tau(\nu) \models (\exists s'').Do(\delta_1, s, s'') \wedge Do(\delta_2, s'', s')$.

4. Suppose δ is $\delta_1 \mid \delta_2$. Then

   $$Do(\delta, s, s') \stackrel{def}{=} Do(\delta_1, s, s') \vee Do(\delta_2, s, s').$$

   The proof is immediate by induction.

5. Suppose δ is $(\pi\ x)\gamma$. Then x must be a variable of sort *action* or *object* and

   $$Do(\delta, s, s') \stackrel{def}{=} (\exists x) Do(\gamma, s, s').$$

   The proof is immediate by induction, and the fact that the *action* and *object* domains of $\mathbb{S}$ and $\mathbb{S}_{pr}$ are the same.

6. Suppose δ is the procedure call $P(t_1, \ldots, t_n)$, where P is a predicate variable of arity $n+2$ whose first n arguments are not of sort *situation*, and whose last two arguments are of sort *situation*. Then

   $$Do(\delta, s, s') \stackrel{def}{=} P(t_1[s], \ldots, t_n[s], s, s').$$

[13] Recall that τ is the mapping from the domain of $\mathbb{S}_{adj}$ onto that of $\mathbb{S}_{pr}$, as described in Definition 5.3. Here, $\tau(\nu)$ is that variable assignment assigning $\tau(d)$ to an individual variable whenever ν assigns d to that variable, where d is an element of the domain of $\mathbb{S}_{adj}$. Moreover, whenever ν assigns $(d_1, \ldots, d_n)$ to an n-ary predicate variable, $\tau(\nu)$ assigns $(\tau(d_1), \ldots, \tau(d_n))$ to that predicate variable.

The result follows from the fact that, by virtue of its definition, τ preserves all functions between $\mathbb{S}_{adj}$ and $\mathbb{S}_{pr}$ (Remark 2).

7. Suppose δ is

   proc $P_1\ (\vec{v}_1)\ \delta_1$ **endProc** ; $\cdots$; **proc** $P_n\ (\vec{v}_n)\ \delta_n$ **endProc** ; δ_0

   The proof here parallels the corresponding proof in Lemma 6.1. Assume, where ν assigns elements from the domain of $\mathbb{S}_{adj}$ to the individual and predicate variables of $\mathcal{L}_{sitcalc}$, that $\mathbb{S}_{adj}, \nu \models Do(\delta, s, s')$. Then,

$$\begin{aligned} \mathbb{S}_{adj}, \nu \models (\forall P_1, \ldots, P_n).[\bigwedge_{i=1}^{n} (\forall s_1, s_2, \vec{v}_i).Do(\delta_i, s_1, s_2) \supset P_i(\vec{v}_i, s_1, s_2)] \\ \supset Do(\delta_0, s, s'). \end{aligned} \tag{24}$$

   For $i = 1, \ldots, n$, let the predicate variable P_i have arity $k_i + 2$, and let $\mathbf{P}_i \subseteq (action \cup object)^{k_i} \times Sit_{pr} \times Sit_{pr}$ be any fixed relation over the domain of $\mathbb{S}_{pr}$, where Sit_{pr} is the *situation* domain of $\mathbb{S}_{pr}$. We must prove that

$$\mathbb{S}_{pr}, \tau(\nu) \models [\bigwedge_{i=1}^{n} (\forall s_1, s_2, \vec{v}_i).Do(\delta_i, s_1, s_2) \supset \mathbf{P}_i(\vec{v}_i, s_1, s_2)] \supset Do(\delta_0, s, s').$$

   So, for $i = 1, \ldots, n$, assume that

$$\mathbb{S}_{pr}, \tau(\nu) \models (\forall s_1, s_2, \vec{v}_i).Do(\delta_i, s_1, s_2) \supset \mathbf{P}_i(\vec{v}_i, s_1, s_2). \tag{25}$$

   We must prove that

$$\mathbb{S}_{pr}, \tau(\nu) \models Do(\delta_0, s, s'). \tag{26}$$

   Let $\mathbf{R}_i = \tau^{-1}(\mathbf{P}_i)$ We prove that, for $i = 1, \ldots, n$

$$\mathbb{S}_{adj}, \nu \models (\forall s_1, s_2, \vec{v}_i).Do(\delta_i, s_1, s_2) \supset \mathbf{R}_i(\vec{v}_i, s_1, s_2). \tag{27}$$

   From this, and (24), it will follow that $\mathbb{S}_{adj}, \nu \models Do(\delta_0, s, s')$, from which, by induction hypothesis, (26) will follow.

   Proof of (27): Let μ be a variable assignment that is just like ν except that it assigns elements from the domain of $\mathbb{S}_{adj}$ to the individual variables $\vec{v}_i, s_1, s_2$. Then we must prove that for any such μ,

   $\mathbb{S}_{adj}, \mu \models Do(\delta_i, s_1, s_2) \supset \mathbf{R}_i(\vec{v}_i, s_1, s_2)$.

   Assume that $\mathbb{S}_{adj}, \mu \models Do(\delta_i, s_1, s_2)$. We prove that $\mathbb{S}_{adj}, \mu \models \mathbf{R}_i(\vec{v}_i, s_1, s_2)$. By induction hypothesis, $\mathbb{S}_{pr}, \tau(\mu) \models Do(\delta_i, s_1, s_2)$. By (25), $\mathbb{S}_{pr}, \tau(\mu) \models \mathbf{P}_i(\vec{v}_i, s_1, s_2)$. By the definition of $\mathbf{R}_i$, $\mathbb{S}_{adj}, \mu \models \mathbf{R}_i(\vec{v}_i, s_1, s_2)$.

■

Corollary 6.2 *Suppose that $\mathbb{S}$ is a structure for $\mathcal{L}_{sitcalc}$. Then for every proper GOLOG program δ,*

$$\mathbb{S}_{adj} \models (\exists s) Do(\delta, S_0, s) \quad \Rightarrow \quad \mathbb{S}_{pr} \models (\exists s) Do(\delta, S_0, s).$$

Theorem 5 (Σ-Elimination Theorem for GOLOG)
Suppose that $\mathcal{D} = \Sigma \cup \mathcal{D}_{ss} \cup \mathcal{D}_{ap} \cup \mathcal{D}_{una} \cup \mathcal{D}_{S_0}$ is a basic action theory and that δ is a proper GOLOG program. Then,

$$\mathcal{D} \models (\exists s) Do(\delta, S_0, s) \quad \textit{iff} \quad \mathcal{D}_{ss} \cup \mathcal{D}_{ap} \cup \mathcal{D}_{una} \cup \mathcal{D}_{S_0} \models (\exists s) Do(\delta, S_0, s).$$

Proof: Suppose $\mathcal{D} \models (\exists s) Do(\delta, S_0, s)$, but $\mathcal{D}_{ss} \cup \mathcal{D}_{ap} \cup \mathcal{D}_{una} \cup \mathcal{D}_{S_0} \not\models (\exists s) Do(\delta, S_0, s)$. Then there is a model $\mathbb{M}$ of $\mathcal{D}_{ss} \cup \mathcal{D}_{ap} \cup \mathcal{D}_{una} \cup \mathcal{D}_{S_0}$ that satisfies $\neg(\exists s) Do(\delta, S_0, s)$. By Corollary 6.1 $\mathbb{M}_{pr}$ satisfies $\neg(\exists s) Do(\delta, S_0, s)$. By Corollary 6.2, $\mathbb{M}_{adj}$ satisfies $\neg(\exists s) Do(\delta, S_0, s)$. By Lemma 5.4, $\mathbb{M}_{adj}$ is a model of $\mathcal{D}$, contradiction. ■

The above theorem is the principal result of this section. It guarantees that the foundational axioms for the situation calculus may be safely ignored for the purposes of evaluating GOLOG programs. Of course, these axioms may well be needed for other purposes:

1. *Proving partial correctness properties:* Provided the initial situation satisfies property ϕ, then whenever program δ terminates, it does so in a situation satisfying ψ.

   $(\forall s, s').\phi(s) \wedge Do(\delta, s, s') \supset \psi(s').$

2. *Proving termination:* Provided the initial situation satisfies property ϕ, then program δ terminates.

   $(\forall s).\phi(s) \supset (\exists s') Do(\delta, s, s').$

Such sentences do not have the syntactic form required by the Σ-Elimination Theorem.

7 Discussion

All the results of this paper are concerned with simplifying the entailment problem for basic theories of action,[14] with implications for the implementation of dynamical systems. For example, Green's [11] classical formulation of planning in AI is as a deduction problem: To obtain a plan to achieve a goal G, establish that $\mathcal{D} \models (\exists s) G(s)$. Any binding for s obtained as a side effect of a proof is a plan for G. Since $(\exists s) G(s)$ is an $\exists s$-sentence, the Σ-Reduction Theorem informs us that for the purposes of planning, various foundational axioms for the situation calculus are not required. Similarly, the projection problem arises frequently in AI, and Theorem 3 assures us that under suitable conditions, regression can solve this problem, and moreover, only the initial database and unique names axioms for actions are necessary for the theorem-proving component of this calculation. Finally, the Σ-Elimination Theorem

[14] Even the relative satisfiability of basic action theories (Theorem 1) can be viewed this way: $\mathcal{D} \models false$ iff $\mathcal{D}_{S_0} \cup \mathcal{D}_{una} \models false$.

for GOLOG informs us that the foundational axioms for the situation calculus need not be considered in implementing interpreters for this programming language.

It would be natural to try to obtain similar results for other classes of problems. One such example that is also of some theoretical and practical interest concerns CONGOLOG [9], a very rich extension of GOLOG that supports concurrent execution with prioritized interrupts and exogenous actions. Is there a Σ-elimination theorem for this programming language?

Regression plays a central computational role in the situation calculus; presumeably it also does so for the many other approaches to modelling dynamics in the artificial intelligence literature (e.g. the families of action languages [7], temporal logic [24], features and fluents [36], the event calculus and its relatives [38, 15, 1]). For suitable analogues of our regressable sentences, can one prove soundness and completeness results for regression in these languages?

References

[1] J.F. Allen. Towards a general theory of action and time. *Artificial Intelligence*, 23(2):123–154, 1984.

[2] L. Bertossi, M. Arenas, and C. Ferretti. SCDBR: An automated reasoner for specifications of database updates. *Journal of Intelligent Information Systems*, 1997. To appear.

[3] W. Burgard, A.B. Cremers, D. Fox, D. Hähnel, G. Lakemeyer, D. Schultz, W. Steiner, and S. Thrun. The interactive museum tour-guide robot. In *Proceedings of the National Conference on Artificial Intelligence (AAAI'98)*, pages 11–18. AAAI Press/MIT Press, 1998.

[4] E. Davis. *Representations of Commonsense Knowledge.* Morgan Kaufmann Publishers, San Francisco, CA, 1990.

[5] H.B. Enderton. *A Mathematical Introduction to Logic.* Academic Press, 1972.

[6] J. Funge. *Making Them Behave: Cognitive Models for Computer Animation.* PhD thesis, Department of Computer Science, University of Toronto, 1998.

[7] M. Gelfond and V. Lifschitz. Action languages. *Linköping Electronic Articles in Computer and Information Sciences*, 3, 1998. www.ep.liu.se/ea/cis/1998/016/.

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

[9] G. De Giacomo, Y. Lespérance, and H.J. Levesque. Reasoning about concurrent execution, prioritized interrupts, and exogenous actions in the situation calculus. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 1221–1226, Nagoya, Japan, 1997.

[10] R. Goldblatt. *Logics of Time and Computation.* CSLI Lecture Notes No. 7. Center for the Study of Language and Information, Stanford Univers ity, Stanford, CA, 2nd edition, 1987.

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

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

[13] S. Hanks and D. McDermott. Default reasoning, nonmonotonic logics, and the frame problem. In *Proceedings of the National Conference on Artificial Intelligence (AAAI'86)*, pages 328–333, 1986.

[14] M. Jenkin, Y. Lespérance, H.J. Levesque, F. Lin, J. Lloyd, D. Marcu, R. Reiter, R.B. Scherl, and K. Tam. A logical approach to portable high-level robot programming. In *Proceedings of the Tenth Australian Joint Conference on Artificial Intelligence (AI'97)*, Perth, Australia, 1997. Invited paper.

[15] R.A. Kowalski and M.J. Sergot. A logic-based calculus of events. *New Generation Computing*, 4:267, 1986.

[16] Y. Lespérance, H.J. Levesque, and R. Reiter. A situation calculus approach to modeling and programming agents. In A. Rao and M. Wooldridge, editors, *Foundations and Theories of Rational Agency*, 1997. In press.

[17] H.J. Levesque. What is planning in the presence of sensing? In *Proceedings of the National Conference on Artificial Intelligence (AAAI'96)*, pages 1139–1146, 1996.

[18] H.J. Levesque, R. Reiter, Y. Lespérance, F. Lin, and R. Scherl. GOLOG: a logic programming language for dynamic domains. *J. of Logic Programming, Special Issue on Actions*, 31(1-3):59–83, 1997.

[19] V. Lifschitz. Toward a metatheory of action. In J. Allen, R. Fikes, and E. Sandewall, editors, *Proceedings of the Second International Conference on Principles of Knowledge Representation and Reasoning (KR'91)*, pages 376–386, Los Altos, CA, 1991. Morgan Kaufmann Publishers, San Francisco, CA.

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

[21] F. Lin and R. Reiter. How to progress a database. *Artificial Intelligence*, 92:131–167, 1997.

[22] J. McCarthy. Situations, actions and causal laws. Technical report, Stanford University, 1963. Reprinted in Semantic Information Processing (M. Minsky ed.), MIT Press, Cambridge, Mass., 1968, pp. 410-417.

[23] J. McCarthy. Epistemological problems of artificial intelligence. In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, pages 1038–1044, Cambridge, MA, 1977.

[24] D. McDermott. Non-monotonic logic II. *J.ACM*, 29:33–57, 1982.

[25] S. A. McIlraith. *Towards a Formal Account of Diagnostic Problem Solving*. PhD thesis, Department of Computer Science, University of Toronto, Toronto, Ontario, Canada, 1997.

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

[27] 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, San Francisco, CA, 1989.

[28] J.A. Pinto. *Temporal Reasoning in the Situation Calculus.* PhD thesis, University of Toronto, Department of Computer Science, 1994.

[29] Javier Pinto. Occurrences and Narratives as Constraints in the Branching Structure of the Situation Calculus. Submitted to the Journal of Logic and Computation
URL = ftp://lyrcc.ing.puc.cl/pub/jpinto/jlc.ps.gz.

[30] R. Reiter. *Knowledge in Action: Logical Foundations for Describing and Implementing Dynamical Systems.* In preparation. Draft available at http://www.cs.toronto.edu/~cogrobo/.

[31] 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.

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

[33] R. Reiter. On specifying database updates. *Journal of Logic Programming*, 25:25–91, 1995.

[34] R. Reiter. Natural actions, concurrency and continuous time in the situation calculus. In L.C. Aiello, J. Doyle, and S.C. Shapiro, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Fifth International Conference (KR'96)*, pages 2–13. Morgan Kaufmann Publishers, San Francisco, CA, 1996.

[35] R. Reiter. Sequential, temporal GOLOG. In A.G. Cohn and L.K. Schubert, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Sixth International Conference (KR'98)*, pages 547–556. Morgan Kaufmann Publishers, San Francisco, CA, 1998.

[36] E. Sandewall. *Features and Fluents: The Representation of Knowledge about Dynamical Systems.* Oxford University Press, 1994.

[37] 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.

[38] M.P. Shanahan. *Solving the Frame Problem: A Mathematical Investigation of the Common Sense Law of Inertia.* MIT Press, 1997.

[39] J.E. Stoy. *Denotational Semantics.* MIT Press, 1977.

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