Linköping Electronic Articles in
Computer and Information Science
Vol. 3(1998): nr 18

Foundations for the Situation Calculus

Hector Levesque, Fiora Pirri, and Ray Reiter

Linköping University Electronic Press
Linköping, Sweden

http://www.ep.liu.se/ea/cis/1998/018/

Published on December 22, 1998 by
Linköping University Electronic Press
581 83 Linköping, Sweden

Linköping Electronic Articles in
Computer and Information Science
ISSN 1401-9841
Series editor: Erik Sandewall

Typeset by the author using LaTeX
Formatted using étendu *style*

Recommended citation:
<Author>. <Title>. Linköping Electronic Articles in Computer and Information Science, Vol. 3(1998): nr 18. http://www.ep.liu.se/ea/cis/1998/018/. *December 22, 1998.*

This URL will also contain a link to the author's home page.

Abstract

This article gives the logical foundations for the situations-as-histories variant of the situation calculus, focusing on the following items:

- The language of the situation calculus.
- Foundational axioms for the domain of situations.
- Axioms for an underlying domain theory.
- The syntax and semantics of the logic programming language GOLOG.
- Axioms for knowledge and sensing actions.
- Essential metatheoretic results about the situation calculus.

Authors' addresses

Hector Levesque
Department of Computer Science
University of Toronto
Toronto, Canada M5S 1A4
`hector@ai.toronto.edu`

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`

1 Introduction

The situation calculus (McCarthy [17]) has long been a staple of AI research, but only recently have there been attempts to carefully axiomatize it. The variant described here has evolved in response to the needs of the Cognitive Robotics Project at the University of Toronto, and its foundations are now sufficiently developed to be gathered in one place. That is what this paper sets out to do.

The principal intuition captured by our axioms is that situations are histories – finite sequences of primitive actions – and we provide a binary constructor $do(a, s)$ denoting the action sequence obtained from the history s by adding action a to it. Other intuitions are certainly possible about the nature of situations. McCarthy and Hayes [19] saw them as "snapshots" of a world. For the purposes of representing narratives, McCarthy and Costello [18] wish to express that any number of actions may occur between a situation and its "successor", and therefore they do not appeal to a constructor like do.

The axioms and supporting metatheory presented in this paper have provided foundations for a wide range of practical work in modeling dynamical systems, including planning (Green [6], Levesque [14]), control (Levesque et al [15], De Giacomo et al [4]), simulation (Kelley [10]), database updates (Reiter [27], Bertossi et al [2]), diagnosis of dynamical systems (McIlraith [20]), and agent programming and robotics (Jenkin et al [9], Lespérance et al [13], Burgard et al [3], Shapiro et al [31], Reiter [29]).

2 The Language of the Situation Calculus

The following description is taken from Pirri and Reiter [24]. $\mathcal{L}_{sitcalc}$ 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 $do : action \times situation \rightarrow situation$. 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 $(action \cup object)^n \rightarrow object$. 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 $(action \cup object)^n \rightarrow action$. 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*.

3 Foundational Axioms for Situations

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. To formalize these desiderata, we adopt the following four foundational axioms for the situation calculus, taken from [24].[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 [26].

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]

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]

3.1 Time, Processes and Concurrency in the Situation Calculus

As presented above, the foundational axioms for situations provide a purely qualitative notion of time whose only temporal concept is sequential action occurrence: An action occurs *before* or *after* another within a situation. There is no way of expressing that an action occurs *at* a particular time, or that two or more actions occur concurrently. While we do not present them here, it is possible to extend these axioms for situations to accommodate time and concurrency. See, for example, Pinto [23] and Reiter [28], and for a discussion of how to model processes in the situation calculus using interleaving concurrency, see Reiter [29].

[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 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 [27] and others. These earlier axiomatizations for the situation calculus are all derivable from the four axioms given here.

4 Domain Axioms and Basic Theories of Actions

Our concern here is 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 4.1 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.

Definition 4.2 Action Precondition Axiom
An action precondition axiom of $\mathcal{L}_{sitcalc}$ is a sentence of the form:

$$Poss(A(x_1, \ldots, x_n), s) \equiv \Pi_A(x_1, \ldots, x_n, s),$$

where A is an n-ary action function symbol, and $\Pi_A(x_1, \cdots, x_n, s)$ 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.

Definition 4.3 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:

$$F(x_1, \ldots, x_n, do(a, s)) \equiv \Phi_F(x_1, \ldots, x_n, a, s), \tag{5}$$

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 [25], is:

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

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:

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

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 [22], Haas [7] and Schubert [30], Reiter [25] 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.

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,

[4] This solution does not take ramification constraints into account, but see [16, 20, 23] for possible ways to do this, while preserving the successor state axiom approach.

- 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 $\mathcal{L}_{sitcalc}$.
- $\mathcal{D}_{S_0}$ 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 4.4 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.

5 Metatheory for the Situation Calculus

Here we present some fundamental logical properties of basic action theories. These are described more fully, with accompanying proofs, in [24]

5.1 Relative Satisfiability

Basic action theories enjoy an important relative satisfiability property:

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

5.2 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. The main result along these lines is the following:

Theorem 2 **(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. Suppose further that* ψ *is a first order sentence of* $\mathcal{L}_{sitcalc}$ *whose prenex normal form has the form*

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

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

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

   Here, $\mathcal{D} - \{Induction\}$ *is* $\mathcal{D}$ *without the second order induction axiom.*

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

   $$\mathcal{D} \models \psi \quad \textit{iff} \quad \Sigma_= \cup \mathcal{D}_{ss} \cup \mathcal{D}_{ap} \cup \mathcal{D}_{una} \cup \mathcal{D}_{S_0} \models \psi.$$

   Here, $\Sigma_=$ *consists of the foundational axiom (1) together with the following sentence:*

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

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

   $$\mathcal{D} \models \psi \quad \textit{iff} \quad \mathcal{D}_{ss} \cup \mathcal{D}_{ap} \cup \mathcal{D}_{una} \cup \mathcal{D}_{S_0} \models \psi.$$

5.3 Regression

Regression is perhaps the single most important theorem-proving mechanism for the situation calculus; it provides a sytematic way to establish that a basic action theory entails a so-called *regressable sentence*.

Definition 5.1 The Regressable Sentences. A regressable sentence of the situation calculus is a first order sentence W of $\mathcal{L}_{sitcalc}$ such that W does not quantify over situations, and such that 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 $\mathcal{L}_{sitcalc}$.

The essence of a regressable sentence 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 sentence 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.

The intuition underlying regression is this: Suppose we want to prove that a regressable 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.

The full definition of the regression operator $\mathcal{R}$ is rather lengthy, mainly because functional fluents introduce certain complications, and we omit it here. See [24] for details. The principal result for regression is the following:

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 result 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. Regression is the central computational mechanism underlying implementations of the logic programming language GOLOG, and we now turn to this.

6 GOLOG

GOLOG is a situation calculus-based logic programming language for implementing complex behaviours for dynamical systems. The language was first introduced in [12, 15], and was described there without pinning down all its formal details. Here, we present the syntax and logical foundations for the language, most of which is taken from Pirri and Reiter [24].

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:

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

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:*

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

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 [5]. 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 [32].

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. However, GOLOG enjoys a strong Σ-Reduction property that somewhat simplifies this task, as we now describe.

6.4 Σ-Elimination for GOLOG

As defined above, a GOLOG program is evaluated relative to a basic action theory $\mathcal{D}$. One can prove [24] that for this purpose, the foundational axioms for situations contained in $\mathcal{D}$ are not needed:

Theorem 4 (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,

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

7 Sensing and Knowledge

To talk about knowledge explicitly in the language of the situation calculus, we follow Moore [21] and suppose that there are situations other than S_0 and its successors that serve as *epistemic alternatives* to the "real" situations. A fluent is considered to be known in a situation s if it holds in all the epistemic alternatives to s. To allow knowledge to change as the result of performing actions, we imagine that certain actions return *sensing results* (here taken to be binary). What is known in a situation s will be fully determined by what was known initially, the actions that led to situation s, and the sensing results of those actions. The material in this section is adapted from Lakemeyer and Levesque [11].

The language of the epistemic situation calculus $\mathcal{L}_e$ is $\mathcal{L}_{sitcalc}$ augmented by two new distinguished symbols: K_0, a unary predicate over situations, and $SF(a, s)$, a binary predicate over actions and situations. Informally, $K_0(s)$ holds when s is an epistemic alternative to S_0, and $SF(a, s)$ holds when action a returns binary sensing result 1 in situation s.

The foundational axioms for $\mathcal{L}_e$ describe the augmented set of situations. First, for any situation term σ, let $Init(\sigma)$ be the abbreviation

$$Init(\sigma) \stackrel{def}{=} (\forall s, a).\ \sigma \neq do(a, s).$$

where a and s do not appear in σ. The first axiom is

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

just as it was in Σ, and

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

which is similar to its counterpart in Σ, stating that all situations are successors of some initial one. Next, we need axioms that define the $\sqsubset$ relation:

$$Init(s') \supset \neg s \sqsubset s',$$

which is an analogue of the one in Σ, and

$$s \sqsubset do(a, s') \equiv s \sqsubseteq s',$$

which is unchanged. Next, we need to say that S_0 and K_0 are initial situations:

$$Init(S_0) \ \wedge \ (\forall s).K_0(s) \supset Init(s).$$

Finally, we need to characterize the set of initial situations. To ensure that nothing need be known about the fluents initially, we insist that there be initial situations (to serve as possible epistemic alternatives) for all combinations of fluent values. To keep things simple here, let us assume that there are only finitely many relational fluents $F_1, \ldots, F_n \in \mathcal{L}_e$, and no functional fluents. We then use the following abbreviation:

$$\begin{array}{l} FV(P_1, \ldots, P_n, P_{n+1}, P_{n+2}, s) \stackrel{def}{=} \\ \quad \bigwedge_{i=1}^{n} (\forall \vec{x}_i)[F_i(\vec{x}_i, s) \equiv P_i(\vec{x}_i)] \\ \qquad \wedge \ (\forall a)[Poss(a, s) \equiv P_{n+1}(a)] \ \wedge \ (\forall a)[SF(a, s) \equiv P_{n+2}(a)]. \end{array}$$

The next foundational axiom is then

$$(\forall \vec{P})(\exists s). \ Init(s) \wedge FV(\vec{P}, s).$$

This is a variant of the existence of situations axiom first introduced by Baker [1]. For some applications it will be desirable to have additional initial situations where actions produce arbitrary effects [11]. Here, we simply assume that actions satisfy the successor state axioms in all epistemic alternatives, and hence we state that there are no additional initial situations. So the final foundational axiom is

$$\begin{array}{l} (\forall s_1, s_2). \ Init(s_1) \wedge Init(s_2) \wedge (s_1 \neq s_2) \ \supset \\ \qquad (\exists \vec{P}).FV(\vec{P}, s_1) \wedge \neg FV(\vec{P}, s_2). \end{array}$$

We refer to these seven foundational axioms as Σ_e.

To characterize what is known, we need two new categories of axioms:

Definition 7.1 Sensed Fluent Axiom
A sensed fluent axiom of $\mathcal{L}_e$ for an action A is a sentence of the form:

$$SF(A(x_1, \ldots, x_n), s) \equiv \Theta_A(x_1, \ldots, x_n, s),$$

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

These axioms are used to characterize the conditions under which the action returns the binary value 1. For example, in a 1-dimensional robotics domain, we might have

$$SF(checkBumper, s) \equiv (\exists x). \ robotPos(x, s) \wedge x < 3.5$$

as a way of saying that a bumper sensor on the robot returns 1 when the robot gets within 3.5 units of a wall located at the origin. Actions that do not involve any sensing can be assumed to always return 1, and hence the Θ would be $TRUE$.

Definition 7.2 Initial Knowledge Axiom
An initial knowledge axiom of $\mathcal{L}_e$ is a sentence of the form:

$$K_0(s) \equiv \Theta(s),$$

where $\Theta(s)$ is a formula that is uniform in s, and whose only free variable is s.

Axioms of this type are used to characterize the epistemic alternatives to S_0. For example, in the 1-dimensional robotics world, we might have

$$K_0(s) \equiv (\exists x).\ x < 10 \wedge (\forall y).\ robotPos(y,s) \equiv x = y.$$

as a way of saying that all the robot knows initially is that it is no further than 10 units away from the wall.

Given these, a *knowledge-action theory* $\mathcal{D}_e$ is of the following form:

$$\mathcal{D}_e = \Sigma_e \cup \mathcal{D}_{ss} \cup \mathcal{D}_{ap} \cup \mathcal{D}_{una} \cup \mathcal{D}_{S_0} \cup \mathcal{D}_{sf} \cup \mathcal{D}_{K_0}$$

where, Σ_e is the new set of foundational axioms, $\mathcal{D}_{ss}$, $\mathcal{D}_{ap}$, $\mathcal{D}_{una}$, and $\mathcal{D}_{S_0}$ are as before, $\mathcal{D}_{sf}$ is a set of sensed fluent axioms, one for each action function symbol, and $\mathcal{D}_{K_0}$ is an initial knowledge axiom.

With such a theory in hand, we can now explain what it means to know a formula in an arbitrary situation by using two abbreviations. First, we have

$$K(\sigma_1, \sigma_2) \stackrel{def}{=} (\forall P)\ [\ldots \supset P(\sigma_1, \sigma_2)]$$

where the ellipsis stands for the universal closure of

$$\begin{array}{l} \{Init(s_2) \quad \wedge \quad K_0(s_1) \supset P(s_1, s_2)\} \quad \wedge \\ \quad \{P(s_1, s_2) \quad \wedge \quad [Poss(a, s_1) \equiv Poss(a, s_2)] \quad \wedge \\ \quad\quad [SF(a, s_1) \equiv SF(a, s_2)] \supset P(do(a, s_1), do(a, s_2))\}. \end{array}$$

The formula $K(s', s)$ can be read as saying that s' is an epistemic alternative to situation s. It is not hard to show that as a binary relation, K is required to be transitive and Euclidean:

Theorem 5 *The closure of the following two formulas is logically valid:*

1. $K(s_2, s_1) \wedge K(s_3, s_2) \supset K(s_3, s_1)$
2. $K(s_2, s_1) \wedge K(s_3, s_1) \supset K(s_3, s_2)$

With respect to knowledge, this means that we are imagining an agent that has positive and negative introspection [8]. The above definition of K also inductively specifies the epistemic alternatives after performing any sequence of actions:

Theorem 6 *The closure of the following two formulas is logically valid:*

1. $K(s', S_0) \equiv K_0(s')$;
2. $K(s', do(a, s)) \equiv (\exists s'').\ s' = do(a, s'') \wedge K(s'', s) \wedge$
$[Poss(a, s'') \equiv Poss(a, s)] \wedge [SF(a, s'') \equiv SF(a, s)]$.

So for an initial situation like S_0, s' is an epistemic alternative only when $K_0(s')$ holds; for a non-initial situation $do(a,s)$, an epistemic alternative is of the form $do(a,s'')$ where s'' is an alternative to s, and where s and s'' agree on $Poss$ and SF.

If we were to consider the formula $K(—,s)$ as a fluent, we can see that the second part of the above theorem amounts to a successor state theorem for K, as it shows how to obtain the value of $K(—,do(a,s))$ as a function of $K(—,s)$ and other properties of a and s. Because we define knowledge in terms of K below, this means we have provided a solution to the frame problem for knowledge: a complete specification of how knowledge changes as the result of performing any action.

Finally, we have the abbreviation

$$Knows(\phi,\sigma) \stackrel{def}{=} (\forall s).\ K(s,\sigma) \supset \phi[s]$$

where ϕ is a situation-suppressed formula and σ is any term of sort *situation*. This says that ϕ is known in situation σ if it comes out true at all epistemic alternatives to σ. Here for simplicity, we are requiring ϕ to be situation-suppressed; a more elaborate account would allow us to express knowledge about knowledge and other formulas where the ϕ mentions situations.

To see how this would work in practice, imagine that we have an action theory $\mathcal{D}_e$ for the 1-dimensional robotics world that includes the two axioms above and

$$(\forall z).\ robotPos(z,S_0) \equiv z = 4.0$$

saying that the actual position of the robot is 4.0 units away from the wall, as well as a successor state axiom for $robotPos$ saying that it is only affected by a $move$ action:

$$(\forall z).\ robotPos(z,do(move(x),s)) \equiv robotPos(z+x,s).$$

Then, letting $WallClose(\sigma)$ stand for the abbreviation

$$(\exists x).\ x < 3.5 \wedge (\forall y).\ robotPos(y,\sigma) \equiv x = y.$$

we get the following as logical consequences of $\mathcal{D}_e$:

$$\neg Knows(WallClose,S_0)$$
$$\neg Knows(\neg WallClose,S_0)$$
$$\neg Knows(WallClose,do(move(-1),S_0))$$
$$\neg Knows(\neg WallClose,do(move(-1),S_0))$$
$$Knows(WallClose,do(checkBumper,do(move(-1),S_0)))$$
$$Knows(WallClose,do(move(-1),do(checkBumper,do(move(-1),S_0))))$$
$$\neg Knows(WallClose,do(move(1),do(checkBumper,do(move(-1),S_0))))$$
$$\neg Knows(\neg WallClose,do(move(1),do(checkBumper,do(move(-1),S_0)))).$$

Acknowledgments

As described here, the formal foundations of the situation calculus have evolved over a number of years, and we gratefully acknowledge the contributions of the present, emeritus and honorary members of the University of Toronto Cognitive Robotics Group: Alfredo Gabaldon, Giuseppe De Giacomo, Todd Kelley, Sam Kaufman, Iluju Kiringai, Gerhard Lakemeyer, Yves Lespérance, Fangzhen Lin, Jeff Lloyd, Sheila McIlraith, Daniel Marcu, Javier Pinto, Shane Ruman, Richard Scherl, Steven Shapiro, Misha

Soutchanski and Eugenia Ternovskaia. We wish also to thank our research sponsors: Agenzia Spaziale Italiana, the National Science and Engineering Research Council of Canada, the Canadian Institute for Robotics and Intelligent Systems, and the Centre for Information Technology of Ontario.

References

[1] A. Baker. A simple solution to the Yale shooting problem. In R. Brachman, H.J. Levesque, and R. Reiter, editors, *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning (KR'89)*, pages 11–20. Morgan Kaufmann Publishers, San Francisco, CA, 1989.

[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] 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.

[5] 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.

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

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

[8] J.Y. Halpern and Y.O. Moses. A guide to completeness and complexity for modal logics of knowledge and belief. *Artificial Intelligence*, 54:319–379, 1992.

[9] 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.

[10] T.G. Kelley. Modeling complex systems in the situation calculus: A case study using the Dagstuhl steam boiler problem. 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 26–37. Morgan Kaufmann Publishers, San Francisco, CA, 1996.

[11] G. Lakemeyer and H. Levesque. AOL: a logic of acting, sensing, knowing, and only knowing. In *Proc. of KR-98, Sixth International Conference on Principles of Knowledge Representation and Reasoning*, pages 316–327, June 1998.

[12] Y. Lespérance, H.J. Levesque, F. Lin, D. Marcu, R. Reiter, and R. Scherl. A logical approach to high-level robot programming – a progress report. In *Control of the Physical World by Intelligent Systems, Working Notes of the 1994 AAAI Fall Symp.*, 1994.

[13] 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.

[14] 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.

[15] 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.

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

[17] 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.

[18] J. McCarthy and T. Costello. Combining narratives. In A.G. Cohn and L.K. Schubert, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Sixth International Conference (KR'98)*, pages 48–59. Morgan Kaufmann Publishers, San Francisco, CA, 1998.

[19] J. McCarthy and P. Hayes. Some philosophical problems from the standpoint of artificial intelligence. In B. Meltzer and D. Michie, editors, *Machine Intelligence 4*, pages 463–502. Edinburgh University Press, Edinburgh, Scotland, 1969.

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

[21] R.C. Moore. A formal theory of knowledge and action. In Jerry B. Hobbs and Robert C. Moore, editors, *Formal Theories of the Commonsense World*, chapter 9, pages 319–358. Ablex Publishing Corp., Norwood, New Jersey, 1985.

[22] 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.

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

[24] F. Pirri and R. Reiter. Some contributions to the metatheory of the situation calculus. *Journal of the ACM*, 1999. To appear. http://www.cs.toronto.edu/~cogrobo/.

[25] 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.

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

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

[28] 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.

[29] 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.

[30] 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.

[31] S. Shapiro, Y. Lespérance, and H.J. Levesque. Goals and rational action in the situation calculus – a preliminary report. In *Working Notes of the AAAI Fall Symposium on Rational Agency: Concepts, Theories, Models and Applications*, Cambridge, MA, 1995.

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