Reasoning about Physical Systems with the Situation Calculus

Todd G. Kelley
Department of Computer Science
University of Toronto
Toronto, Canada M5S 1A4
email: tgk@cs.toronto.edu

Abstract

I use the results of (Reiter 1996) to show how the situation calculus can be used to reason about a physical system with continuously varying parameters and concurrent actions. The situation calculus model of a toilet is discussed as an example that is well understood, yet interesting. We present a PROLOG technology simulator for situation calculus models, and describe how to translate a situation calculus axiom into its PROLOG equivalent.

Introduction

The situation calculus, in its latest form, can handle qualitative reasoning about physical systems, including continuous parameters and concurrent actions. Reiter (Reiter 1996) has provided a firm theoretical foundation for modeling physical systems and simulating their behavior.

Inspired by Rieger (Rieger 1985), I have used this situation calculus to model toilet behavior. The toilet seems to be a good example because it is simple enough to be understood by everyone, but complicated enough to be interesting.

Situation calculus

The instantiation of McCarthy's (McCarthy & Hayes 1969) situation calculus language used in this paper is due to Reiter (Reiter 1996), largely influenced by Pinto's (Pinto 1994) work on concurrency and continuous processes.

The Language

The language used in this paper consists of the following ontology:

- a sort *situation*, and a distinguished situation constant symbol S_0.
- a sort *time* ranging over the reals.
- a sort *action* of *simple* actions. All actions are instantaneous, and take a parameter in the last argument position denoting the time of the action's occurrence. Variables a, a', etc. are used for simple actions.
- a sort *concurrent* of concurrent actions which are sets of simple actions. Variables c, c', etc. are used for concurrent actions.
- a function symbol $time : action \to \Re$, where $time(a)$ denotes the time of the action a.
- a function symbol $start : situation \to \Re$, where $start(s)$ denotes the start time of the situation s.
- a function symbol $do : action \times situation \to situation$
- The predicate symbol $Poss$, where $Poss(a, s)$ means that the simple action a is possible in situation s (and similarly for any concurrent action c).
- The predicate symbol $<$, where $s < s'$ means that s' is reachable from s through the execution of a sequence of possible actions (simple or concurrent).
- The foundational axioms for the concurrent situation calculus, provided in (Reiter 1996), which are generalizations of those provided in (Lin & Reiter 1994) and (Reiter 1993) for the nonconcurrent situation calculus. These axioms include unique names axioms for situations, a definition for $<$, a coherency criterion for concurrent actions, and an induction axiom.

Axiomatizing Application Domains

Levesque *et al.* (Levesque *et al.* 1996) list the general types of axioms required to formalize an application domain in the situation calculus. In particular, our axiomatization consists of the following axioms:

- For each *simple* action A, a single action precondition axiom of the form
$$Poss(A(\vec{x}, t), s) \equiv start(s) \leq t \wedge \Phi(\vec{x}, t, s)$$
where $\Phi(\vec{x}, t, s)$ is any first order formula with free variables among $\vec{x}$, t, and s whose only term of sort *situation* is s.
- For each fluent R, a single successor state axiom. The form of a successor state axiom for a relational fluent, F, is
$$Poss(c, s) \supset F(\vec{x}, do(c, s)) \equiv$$
$$\gamma_F^+(\vec{x}, c, s) \vee F(\vec{x}, s) \wedge \neg\gamma_F^-(\vec{x}, c, s)$$

where $\gamma_F^+(\vec{x}, c, s)$ and $\gamma_F^-(\vec{x}, c, s)$ denote the conditions under which c, if performed in s, results in $F(\vec{x}, do(c, s))$ becoming true and false, respectively.

For a functional fluent, f, the form of the successor state axiom is

$$Poss(c, s) \supset f(\vec{x}, do(c, s)) = y \equiv \\ \gamma_f(\vec{x}, y, c, s) \vee y = f(\vec{x}, s) \wedge \neg(\exists y')\gamma_f(\vec{x}, y', c, s)$$

Here, $\gamma_f(\vec{x}, y, c, s)$ is a first order formula whose free variables are among $\vec{x}, y, c, s$.

- Unique names axioms for the primitive actions.
- Axioms describing the initial situation.
- The foundational axioms mentioned in the previous section.

Modeling Toilet Behavior

In this section, I present an axiomatization of a toilet. Because different people have different notions of the workings of a toilet, I first provide an English description of toilet behavior which shall be considered correct enough for the purposes of this paper.

A toilet consists of a tank and a bowl. The tank has a plug at the bottom which is opened by pushing the handle. The handle holds the plug open until the handle is released, at which point the plug either floats if the tank is not empty, or closes if the tank is empty.

When the plug is open, water rushes out of the tank and into the bowl with rate $Q_{t \to b}$. When the tank level drops below full, the supply opens, so that water flows from the supply into the tank at rate $Q_{s \to t}$, and into the bowl at rate $Q_{s \to b}$. Half the new water from the supply goes into the bowl, and half goes into the tank, meaning $Q_{s \to b} = Q_{s \to t}$.

By now, water is rushing into the bowl at a rate of $Q_{t \to b} + Q_{s \to b}$. Since the bowl is full, water exits the waste channel at the same rate. This rate is greater than the threshold of siphoning, Q_{siphon_thres}, and siphoning begins. The siphoning flow rate through the waste channel is $Q_{t \to b} + Q_{s \to b}$ (the flow rate does not change with siphoning).

Eventually, the tank will become empty and the plug will close (if it was released). The flow from the tank to the bowl drops to $Q_{s \to b}$, which is less than the siphoning rate, and the bowl empties. Once the bowl is empty, siphoning must stop.

Water continues to flow into both the bowl and the tank until the tank is full and the supply closes. Since the bowl is not at big as the tank, it will fill up before the tank does.

This scenario can be modeled with the following fluents:

- *supply_open*(s): the supply of incoming water is open
- *siphoning*(s): water is siphoning out of the bowl into the waste channel
- *handle_pushed*(s): the handle is held in the activated position
- *plug_held*(s): the plug at the bottom of the tank is held open
- *plug_floating*(s): the plug at the bottom of the tank is floating, and open
- *plug_closed*(s): the plug at the bottom of the tank is closed
- $flow\_supply\_tank(s) = f$: f, a function of time, is the flow of new water from the supply into the tank in s
- $flow\_supply\_bowl(s) = f$: f, a function of time, is the flow of new water from the supply into the bowl in s
- $flow\_tank\_bowl(s) = f$: f, a function of time, is the flow of water from the tank into the bowl in s
- $flow\_bowl\_channel(s) = f$: f, a function of time, is the flow of water from the bowl into the waste channel in s
- $bowl\_level(s) = f$: f, a function of time, is the bowl water level in s
- $tank\_level(s) = f$: f, a function of time, is the tank water level in s

and the following natural actions:

- *push_handle*(0)
- *release_handle*(0)
- *open_supply*(t)
- *close_supply*(t)
- *close_plug*(t)
- *begin_siphon*(t)
- *end_siphon*(t)
- *bowl_become_full*(t)
- *bowl_become_empty*(t)
- *tank_become_full*(t)
- *tank_become_empty*(t)

Toilet Precondition Axioms

The handle can be pushed only if it is not pushed already.

$$Poss(push\_handle(t), s) \equiv \\ t \geq start(s) \wedge \neg handle\_pushed(s) \qquad (1)$$

The handle can be released only if it is not released already.

$$Poss(release\_handle(t), s) \equiv \\ t \geq start(s) \wedge handle\_pushed(s) \qquad (2)$$

The tank becomes empty if the tank water level is dropping and at 0.

$$Poss(tank\_become\_empty(t), s) \equiv \\ t \geq start(s) \wedge tank\_level(s)(t) = 0 \wedge \\ \frac{d}{dt}(tank\_level(s))(t) < 0 \quad (3)$$

The tank becomes full if the tank water level is increasing and at $full\_tank\_level$.

$$Poss(tank\_become\_full(t), s) \equiv \\ t \geq start(s) \wedge tank\_level(s)(t) = full\_tank\_level \wedge \\ \frac{d}{dt}(tank\_level(s))(t) > 0 \quad (4)$$

The bowl becomes empty if the bowl water level is dropping and at 0.

$$Poss(bowl\_become\_empty(t), s) \equiv \\ t \geq start(s) \wedge bowl\_level(s)(t) = 0 \wedge \\ \frac{d}{dt}(bowl\_level(s))(t) < 0 \quad (5)$$

The bowl becomes full if the bowl water level is increasing and at $full\_bowl\_level$.

$$Poss(bowl\_become\_full(t), s) \equiv \\ t \geq start(s) \wedge bowl\_level(s)(t) = full\_bowl\_level \wedge \\ \frac{d}{dt}(bowl\_level(s))(t) > 0 \quad (6)$$

The supply opens if the tank level is decreasing, and at $full\_tank\_level$.

$$Poss(open\_supply(t), s) \equiv \\ t \geq start(s) \wedge tank\_level(s)(t) = full\_tank\_level \wedge \\ \frac{d}{dt}(tank\_level(s))(t) < 0 \quad (7)$$

The supply closes if the tank level is increasing, and at $full\_tank\_level$.

$$Poss(close\_supply(t), s) \equiv \\ t \geq start(s) \wedge tank\_level(s)(t) = full\_tank\_level \wedge \\ \frac{d}{dt}(tank\_level(s))(t) > 0 \quad (8)$$

The plug at the bottom of the tank closes if it is floating and the tank is empty.

$$Poss(close\_plug(t), s) \equiv \\ t \geq start(s) \wedge plug\_floating(s) \wedge \\ tank\_level(s)(t) = 0 \quad (9)$$

Siphoning begins if siphoning is not currently happening, and the flow through the waste channel is greater than Q_{siphon_thres}.

$$Poss(begin\_siphon(t), s) \equiv \\ t \geq start(s) \wedge \neg siphoning(s) \wedge \\ flow\_bowl\_channel(s)(t) > Q_{siphon\_thres} \quad (10)$$

Siphoning ends if siphoning is currently happening, and the level of water in the bowl is 0.

$$Poss(end\_siphon(t), s) \equiv \\ t \geq start(s) \wedge siphoning(s) \wedge \\ bowl\_level(s)(t) = 0 \quad (11)$$

Toilet Successor State Axioms

The successor state axioms for $handle\_pushed(s)$, $plug\_held(s)$, $siphoning(s)$, and $plug\_floating(s)$ are straightforward, and similar to the one for $supply\_open(s)$:

$$Poss(c, s) \supset supply\_open(do(c, s)) \equiv \\ close\_supply(time(c)) \notin c \wedge supply\_open(s) \vee \\ open\_supply(time(c)) \in c \quad (12)$$

The axiom for $flow\_supply\_bowl(s)$ is similar to that for $flow\_supply\_tank(s)$:

$$Poss(c, s) \supset flow\_supply\_tank(do(c, s)) = f \equiv \\ open\_supply(time(c)) \in c \wedge f(t) = Q_{s \to t} \vee \\ close\_supply(time(c)) \in c \wedge f(t) = 0 \vee \\ open\_supply(t) \notin c \wedge close\_supply(t) \notin c \wedge \\ flow\_supply\_tank(s) = f \quad (13)$$

The successor state axiom for $plug\_closed(s)$ is interesting:

$$Poss(c, s) \supset plug\_closed(do(c, s)) \equiv \\ push\_handle(time(c)) \notin c \wedge \\ [plug\_closed(s) \vee close\_plug(time(c)) \in c] (14)$$

This is a case where the effects of one action (namely, $push\_handle(t)$) on the fluent conflict with what we would intuitively expect to be the effect of another action (namely, $close\_plug(t)$). The same principle arises with the successor state axiom for $flow\_tank\_bowl(s)$, below. See the discussion section for more on this topic.

The successor state axioms for $flow\_tank\_bowl(s)$ and $flow\_bowl\_channel(s)$ are complicated. There are five simple actions that can affect the $flow\_tank\_bowl(s)$ fluent: $open\_supply(t)$, $close\_supply(t)$, $push\_handle(t)$, $close\_plug(t)$, and $tank\_become\_empty(t)$. Because the actions can happen concurrently (except $open\_supply(t)$ and $close\_supply(t)$), there are 24 different concurrent actions that can affect the $flow\_tank\_bowl(s)$ fluent. The situation for $flow\_bowl\_channel(s)$ is worse, since there are seven simple actions that can affect it.

The following axiom is the successor state axiom for the $flow\_tank\_bowl(s)$ fluent, but only part of it is shown explicitly (the dots represent the rest).

$$
\begin{aligned}
&Poss(c,s) \supset flow\_tank\_bowl(do(c,s)) = f \equiv \\
&\quad close\_supply(time(c)) \in c \wedge \\
&\quad open\_supply(time(c)) \notin c \wedge \\
&\quad push\_handle(time(c)) \notin c \wedge \\
&\quad close\_plug(time(c)) \notin c \wedge \\
&\quad tank\_become\_empty(time(c)) \notin c \wedge \\
&\quad [plug\_closed(s) \wedge f = flow\_tank\_bowl(s) \vee \\
&\quad \neg plug\_closed(s) \wedge \\
&\quad [tank\_level(s)(time(c)) = 0 \wedge f(t) = 0 \vee \\
&\quad tank\_level(s)(time(c)) \neq 0 \wedge f(t) = Q_{t \to b}]] \vee \\
&\quad push\_handle(time(c)) \in c \wedge \\
&\quad close\_plug(time(c)) \in c \wedge \\
&\quad tank\_become\_empty(time(c)) \in c \wedge \\
&\quad close\_supply(time(c)) \notin c \wedge \\
&\quad open\_supply(time(c)) \notin c \wedge \\
&\quad [supply\_open(s) \wedge f(t) = Q_{s \to t} \vee \\
&\quad \neg supply\_open(s) \wedge f(t) = 0] \vee \\
&\quad \vdots \\
&\quad \ldots \vee \\
&\quad close\_supply(time(c)) \notin c \wedge \\
&\quad open\_supply(time(c)) \notin c \wedge \\
&\quad push\_handle(time(c)) \notin c \wedge \\
&\quad close\_plug(time(c)) \notin c \wedge \\
&\quad tank\_become\_empty(time(c)) \notin c \wedge \\
&\quad f = flow\_tank\_bowl(s) \qquad (15)
\end{aligned}
$$

This axiom states, among other things, that if the only action in c relevant to the flow of water from the tank to the bowl is $close\_supply(t)$, then the effect of c on the flow depends on the status of the plug at the bottom of the tank. Further, if the plug is closed, closing the supply has no effect on the flow from the tank to the bowl; however, if the plug is open, the effect depends on whether the tank is empty. If the tank is not empty, closing the supply has no effect, but if the tank is empty, closing the supply ceases any flow from the tank to the bowl.

This axiom also shows what happens when $close\_plug(t)$ and $push\_handle(t)$ occur simultaneously. The result is the same as if $close\_plug(t)$ had not happened. In other words, the effect of a $push\_handle(t)$ action supersedes the effect of a simultaneous $close\_plug(t)$ action.

Due to its excessive size, the successor state axiom for $flow\_bowl\_channel(s)$ is omitted.

State Constraints

Reiter's solution to the frame problem (Reiter 1991) relies on the assumption that there are no state constraints. The problem with state constraints is that they can lead to indirect effects which violate the Completeness Assumption on which the solution depends. The following state constraints cannot violate the Completeness Assumption, because we use these state constraints in lieu of successor state axioms to describe the $tank\_level(s)$ and $bowl\_level(s)$ fluents. We could write down successor state axioms that would be equivalent to these state constraints, but they would be much larger.

$$
\begin{aligned}
&Poss(c,s) \supset bowl\_level(do(c,s)) = f \equiv \\
&\quad f(t) = bowl\_level(s)(time(c)) + \\
&\quad \int_{time(c)}^{t} flow\_supply\_bowl(do(c,s)) + \\
&\quad flow\_tank\_bowl(do(c,s)) - \\
&\quad flow\_bowl\_channel(do(c,s))dt \qquad (16)
\end{aligned}
$$

$$
\begin{aligned}
&Poss(c,s) \supset tank\_level(do(c,s)) = f \equiv \\
&\quad f(t) = tank\_level(s)(time(c)) + \\
&\quad \int_{time(c)}^{t} flow\_supply\_tank(do(c,s)) - \\
&\quad flow\_tank\_bowl(do(c,s))dt \qquad (17)
\end{aligned}
$$

Background Knowledge

Although we may not know (or care) what the exact real values of the various flows are, we do have qualitative knowledge about those values:

- $Q_{s \to b} > 0$, $Q_{s \to t} > 0$, $Q_{t \to b} > 0$, $Q_{siphon} > 0$, $Q_{siphon\_thres} > 0$: all flow rates are positive
- $Q_{t \to b} > Q_{siphon\_thres}$: the flow from the tank to the bowl when the plug is open and the tank is not empty is enough to initiate siphoning
- $Q_{siphon\_thres} > Q_{s \to b}$: the flow from the supply to the bowl is not enough to initiate siphoning
- $Q_{t \to b} + Q_{s \to b} = Q_{siphon}$: when the tank is not yet empty, the level of water in the bowl does not change during siphoning
- $Q_{s \to t} < Q_{t \to b}$: the tank will eventually empty if the plug is open, even if the supply is on
- $Q_{siphon} > Q_{s \to b}$: the bowl will eventually empty due to siphoning, even if the supply is on
- $Q_{s \to b} = Q_{s \to t}$: the supply water is divided evenly between tank and bowl
- $Q_{s \to b} + Q_{s \to t} < Q_{siphon}$: the total supply water flow is less than the siphoning flow

Simulation

In this section I discuss a PROLOG technology simulator. The simulator can be used to reason about a concurrent situation calculus model like the axiomatization of a toilet presented in the previous section.

The PROLOG Simulator

A situation calculus model defines a tree of situations emanating from the distinguished situation S_0. Some of the situations in the tree correspond to *legal* situations, and some do not. A legal situation is consistent with the laws of Nature, in that a natural action must occur at the time dictated by natural laws governing the behavior of the system, unless the action is prevented from occurring by an earlier natural action. Reiter (Reiter 1996) defines the $legal(S)$ predicate to formalize this principle:

$$\begin{aligned} & legal(s) \equiv \\ & \quad S_0 \leq s \land \\ & \quad (\forall a, c, s').natural(a) \land Poss(a, s') \land \\ & \quad do(c, s') \leq s \land a \notin c \supset time(c) < time(a). \end{aligned} \tag{18}$$

Here, $\leq$ is the ordering relation defined by the foundational axioms mentioned earlier. The *legal* predicate is instrumental in the implementation of a simulator, as will become clear.

A domain of discourse in which all actions are natural is said to comply with Reiter's (Reiter 1996) Natural World Condition (NWC). This condition assures a deterministic simulation.

Another concept crucial to the implementation of a simulator is the notion of Reiter's (Reiter 1996) Least Natural Time Points:

$$\begin{aligned} & lntp(s, t) \equiv \\ & \quad (\exists a)[natural(a) \land Poss(a, s) \land time(a) = t \land \\ & \quad (\forall a')[natural(a') \land Poss(a', s) \supset \\ & \quad time(a') \geq t]. \end{aligned} \tag{19}$$

Informally, the least natural time point is the earliest time at which any natural action can possibly occur in a situation. The Least Natural Time Point Condition ($LNTPC$) is the following:

$$(\forall s).(\exists a)[natural(a) \land Poss(a, s)] \supset (\exists t) lntp(s, t). \tag{20}$$

Reiter (Reiter 1996) puts this all together and proves:

$$\begin{aligned} & LNTPC \land NWC \supset legal(do(c, s)) \equiv \\ & \quad legal(s) \land Poss(c, s) \land \\ & \quad (\forall a)[a \in c \equiv Poss(a, s) \land lntp(s, time(a))] \end{aligned} \tag{21}$$

Formula 21 is the engine for the simulator. The simulator is a PROLOG procedure that takes a situation term s as an argument (initially S_0), prints its argument, constructs a set of actions c such that

$$(\forall a)[a \in c \equiv Poss(a, s) \land lntp(s, time(a))],$$

and recursively calls itself with $do(c, s)$. In so doing, the simulator follows the path of legal situations (there is only one path of legal situations when all actions are natural), simulating the evolution of the system.

Here is the PROLOG code:

```
what_happens(S) :-
    nl,nl,print(S),
    setof(A,occurrence(A,S),C),
    what_happens(do(C,S)).

occurrence(A,S):-
    natural(A)),
    poss(A,S),
    time(A,T),
    not (
         natural(A_prime),
         poss(A_prime,S),
         time(A_prime,T_prime),
         sign(T-T_prime,pos)
        ).
```

Translating a Model to PROLOG

Clark's completion semantics for logic programming admit a translation from the situation calculus axioms of the toilet model to PROLOG clauses. The procedure is to simply make the implication in the axioms go only one way, and write down the clausal form.

For example, Precondition Axiom 3 becomes

```
%%%% The tank becomes empty if the
%%%% tank level is dropping, and
%%%% the level is 0.
poss(tank_become_empty(T0),S):-
   % tank_level(T,S) = Lt
   tank_level(Lt,T,S),
   % Lt(T) = 0 when T = T0
   solve(Lt=0,T,T0),
   % d/dT Lt(T) < 0
   sign(subst(T0,T,diff(Lt,T)),neg),
   start(S,T_s),
   % if it can happen at T_s,
   % it will
   (T0=T_s; not T0=T_s),
   % start(S) <= T0
   (sign(T0-T_s,pos) ; sign(T0-T_s,zero)).
```

where

- `solve(Eqn,Var,Soln)` asserts that `Soln` solves the equation `Eqn` for `Var`.
- `subst(X,Y,Expr)` is an expression the same as `Expr` with all occurrences of `Y` replaced by `X`.
- `sign(X,S)` asserts that `S` is the sign of expression `X`.
- `diff(Expn,Var)` is the derivative of `Expr` w.r.t. variable `Var`

Successor State Axiom 12 becomes

```
supply_open(do(A,S)):-
    poss(A,S),
    (
      not member(close_supply(T0),A),
      supply_open(S)
    ;
      member(open_supply(T0),A).
    ).
```

The Output

When we issue the query, `what_happens(s0)`, the last state the simulator prints out (just before it fails to find a successor state) is:

```
do([tank_become_full(2*qtb/(qst*qtb-qst^2)),
    close_supply(2*qtb/(qst*qtb-qst^2))],
 do([bowl_become_full(
      (-((2*qsb+qbc)*qtb-qbc*qst))/
       ((qsb^2-qbc*qsb)*qtb+
         (qbc*qsb-qsb^2)*qst))],
  do([bowl_become_empty((-(3*qtb-qst))/
      ((qsb-qbc)*qtb+(qbc-qsb)*qst))],
   do([close_plug(2/(qtb-qst)),
       tank_become_empty(2/(qtb-qst))],
    do([begin_siphon(0), open_supply(0),
        release_handle(0)],
     do([push_handle(0)],s0))))))))
```

Discussion

This paper has uncovered three issues of interest:

Situations can have zero duration: In Reiter's paper (Reiter 1996), he imposes the global constraint,

$$start(s) < start(do(c, s)),$$

which states that no situation has zero duration. There are cases, however, where a situation with zero duration is intuitively desirable, and for that reason, the version of the constraint in this paper has been

$$start(s) \leq start(do(c, s)).$$

A result of the more relaxed constraint is the output

```
do([begin_siphon(0),
        open_supply(0),
        release_handle(0)],
    do([push_handle(0)],s0))))))))
```

which is intuitively correct, since it should not be possible for the supply to open until after the handle has been pushed. However, with this idealized toilet, the supply must open immediately as the handle is pushed.

Actions can cancel out the effects of others: As previously pointed out, when *push_handle* and *close_plug* occur in the same concurrent action, the effect of *push_handle* takes precedence over the effect of *close_plug*. This phenomenon is closely related to the Precondition Interaction Problem of (Pinto 1994), but it is not the same. In particular, successor state axioms—not precondition axioms—are the proper place to deal with it. Consider another example: a ball that is caught just as it bounces. The observed effect is that of the catch action, but the bounce also occurred, since after the catch, the ball is still touching the ground.

Successor state axioms can be large: The cause of this practical (rather than theoretical) problem is that every possible combination of the actions that can affect a fluent must be considered as a separate action. Hence, the number of actions the axiomatizer must consider when writing a successor state axiom grows exponentially with the number of simple actions that can affect the fluent. A naive approach might be to consider every situation with concurrent actions to have only simple actions, some of which occur at the same time. The hope would be that the effects of the simple actions will naturally add up. This approach is overly simple and will not work: consider two cowboys who shoot each other simultaneously. Both cowboys are dead in $do([shoot1(0), shoot2(0)], S_0)$, but $do(shoot1(0), do(shoot2(0), s0))$ is not even a legal situation.

Acknowledgements

I thank Ray Reiter for his comments on an earlier version of this paper. Ray's paper (Reiter 1996) provides a theoretical foundation for this one. I have also benefited from discussions with Javier Pinto.

References

Levesque, H. J.; Reiter, R.; Lespérance, Y.; Lin, F.; and Scherl, R. 1996. GOLOG: A logic programming language for dynamic domains. *Journal of Logic Programming, Special Issue on Reasoning about Action and Change.* To appear.

Lin, F., and Reiter, R. 1994. State constraints revisited. *Journal of Logic and Computation* 4(5):655–678.

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

Pinto, J. A. 1994. *Temporal Reasoning in the Situation Calculus.* Ph.D. Dissertation, University of Toronto, Toronto, Ontario, Canada.

Reiter, R. 1991. The frame problem in the situation calculus: a simple solution (sometimes) and a completeness result for goal regression. In Lifschitz, V., ed., *Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy.* San Diego, CA: Academic Press. 359–380.

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

Reiter, R. 1996. Natural actions, concurrency and continuous time in the situation calcu lus. In *COMMON SENSE '96: the third symposium on logical formalizations of commonsense reasoning.*

Rieger, C. 1985. An organization of knowledge for problem solving and language comprehension. In Brachman, R. J., and Levesque, H. J., eds., *Readings in Knowledge Representation.* Los Altos, CA: Morgan Kaufmann.