Handbook of Knowledge Representation
Edited by F. van Harmelen, V. Lifschitz and B. Porter


DOI: 10.1016/S1574-6526(07)03016-7

# Chapter 16

# Situation Calculus

## Fangzhen Lin

The situation calculus is a logical language for representing changes. It was first introduced by McCarthy in 1963,[1] and described in further details by McCarthy and Hayes [29] in 1969.

The basic concepts in the situation calculus are *situations*, *actions* and *fluents*. Briefly, actions are what make the dynamic world change from one situation to another when performed by agents. Fluents are situation-dependent functions used to describe the effects of actions. There are two kinds of them, *relational* fluents and *functional* fluents. The former have only two values: true or false, while the latter can take a range of values. For instance, one may have a relational fluent called *handempty* which is true in a situation if the robot's hand is not holding anything. We may need a relation like this in a robot domain. One may also have a functional fluent called *battery-level* whose value in a situation is an integer between 0 and 100 denoting the total battery power remaining on one's laptop computer.

According to McCarthy and Hayes [29], a situation is "the complete state of the universe at an instance of time". But for Reiter [34], a situation is the same as its history which is the finite sequence of actions that has been performed since the initial situation $S_0$. We shall discuss Reiter's foundational axioms that make this precise later. Whatever the interpretation, the unique feature of the situation calculus is that situations are first-order objects that can be quantified over. This is what makes the situation calculus a powerful formalism for representing change, and distinguishes it from other formalisms such as dynamic logic [11].

To describe a dynamic domain in the situation calculus, one has to decide on the set of actions available for the agents to perform, and the set of fluents needed to describe the changes these actions will have on the world. For example, consider the classic blocks world where some blocks of equal size can be arranged into a set of towers on a table. The set of actions in this domain depends on what the imaginary agent can do. If we imagine the agent to be a one-handed robot that can be directed to grasp any block that is on the top of a tower, and either add it to the top of another tower or put it down on the table to make a new tower, then we can have the following actions [30]:

[1] In a Stanford Technical Report that was later published as [25].

- *stack*$(x, y)$—put block $x$ on block $y$, provided the robot is holding $x$, and $y$ is clear, i.e. there is no other block on it;
- *unstack*$(x, y)$—pick up block $x$ from block $y$, provided the robot's hand is empty, $x$ is on $y$, and $x$ is clear;
- *putdown*$(x)$—put block $x$ down on the table, provided the robot is holding $x$;
- *pickup*$(x)$—pick up block $x$ from the table, provided the robot's hand is empty, $x$ is on the table and clear.

To describe the effects of these actions, we can use the following relational fluents:

- *handempty*—true in a situation if the robot's hand is empty;
- *holding*$(x)$—true in a situation if the robot's hand is holding block $x$;
- *on*$(x, y)$—true in a situation if block $x$ is on block $y$;
- *ontable*$(x)$—true in a situation if block $x$ is on the table;
- *clear*$(x)$—true in a situation if block $x$ is the top block of a tower, i.e. the robot is not holding it, and there is no other block on it.

So, for example, we can say that for action *stack*$(x, y)$ to be performed in a situation, *holding*$(x)$ and *clear*$(y)$ must be true, and that after *stack*$(x, y)$ is performed, in the resulting new situation, *on*$(x, y)$ and *handempty* will be true, and *holding*$(x)$ and *clear*$(y)$ will no longer be true.

If, however, the agent in this world can move a block from a clear position to another clear position, then we only need the following action:

- *move*$(x, y)$—move block $x$ to position $y$, provided that block $x$ is clear to move, where a position is either a block or the table.

To describe the effects of this action, it suffices to use two fluents *on*$(x, y)$ and *clear*$(x)$: action *move*$(x, y)$ can be performed in a situation if $x \neq$ *table*, *clear*$(x)$, and *clear*$(y)$ are true in the situation, and that after *move*$(x, y)$ is performed, in the resulting new situation, $x$ is no longer where it was but on $y$ now.

To axiomatize dynamic domains like these in the situation calculus, we will need to be a bit more precise about the language.

## 16.1 Axiomatizations

We said that the situation calculus is a logical language for reasoning about change. More precisely, it is a first-order language, sometime enriched with some second-order features. It represents situations and actions as first-order objects that can be quantified over. Thus we can have a first-order sentence saying that among all actions, *putdown*$(x)$ is the only one that can make *ontable*$(x)$ true. We can also have a first-order sentence saying that in any situation, executing different actions will always yield different situations. As we mentioned, being able to quantify over situations makes the situation calculus a very expressive language, and distinguishes it from other formalisms for representing dynamic systems.

As we mentioned, fluents are functions on situations. Of special interest are relational fluents that are either true or false in a situation. Initially, McCarthy and Hayes represented relational fluents as predicates whose last argument is a situation term [29]. For instance, to say that block $x$ is on the table in situation $s$, one would use a binary predicate like *ontable* and write $ontable(x, s)$. This was also the approach taken by Reiter [33, 34]. Later, McCarthy [26, 28] proposed to reify relational fluents as first-order objects as well, and introduced a special binary predicate "$Holds(p, s)$" to express the truth value of a relational fluent $p$ in situation $s$. Here we shall follow McCarthy's later work, and represent relational fluents as first-order objects as well. This allows us to quantify over fluents. But more importantly, it allows us to talk about other properties of fluents like causal relationships among them [15]. One could continue to write formulas like $ontable(x, s)$, which will be taken as a shorthand for $Holds(ontable(x), s)$.

To summarize, the situation calculus is a first-order language with the following special sorts: *situation*, *action*, and *fluent* (for relational fluents). There could be other sorts, some of them domain dependent like *block* for blocks in the blocks world or *loc* for locations in logistics domain, and others domain independent like *truth* for truth values. For now we assume the following special domain independent predicates and functions:

- $Holds(p, s)$—fluent $p$ is true in situation $s$;
- $do(a, s)$—the situation that results from performing action $a$ in situation $s$;
- $Poss(a, s)$—action $a$ is executable in situation $s$.

Other special predicates and functions may be introduced. For instance, to specify Golog programs [12], one can use a ternary predicate called $Do(P, s_1, s_2)$, meaning that $s_2$ is a terminating situation of performing program $P$ in $s_1$. To specify causal relations among fluents, one can use another ternary predicate $Caused(p, v, s)$, meaning that fluent $p$ is caused to have truth value $v$ in situation $s$.

Under these conventions, a relational fluent is represented by a function that does not have a situation argument, and a functional fluent is represented by a function whose last argument is of sort *situation*. For instance, $clear(x)$ is a unary relational fluent. We often write $clear(x, s)$ which as we mentioned earlier, is just a shorthand for $Holds(clear(x), s)$. On the other hand, $color(x, s)$ is a binary functional fluent, and we write axioms about it like

$$color(x, do(paint(x, c), s)) = c.$$

We can now axiomatize our first blocks world domain with the following first-order sentences (all free variables are assumed to be universally quantified):

$$Poss(stack(x, y), s) \equiv holding(x, s) \wedge clear(y, s), \tag{16.1}$$

$$Poss(unstack(x, y), s) \equiv on(x, y, s) \wedge clear(x, s) \wedge handempty(s), \tag{16.2}$$

$$Poss(pickup(x), s) \equiv ontable(x, s) \wedge clear(x, s) \wedge handempty(s), \tag{16.3}$$

$$Poss(putdown(x), s) \equiv holding(x, s), \tag{16.4}$$

$$holding(u, do(stack(x, y), s)) \equiv holding(u, s) \wedge u \neq x, \tag{16.5}$$

$$handempty(do(stack(x, y), s)), \tag{16.6}$$

$$on(u, v, do(stack(x, y), s)) \equiv (u = x \wedge v = y) \vee on(u, v, s), \tag{16.7}$$

$$clear(u, do(stack(x, y), s)) \equiv u = x \vee (clear(u, s) \wedge u \neq y), \tag{16.8}$$

$$ontable(u, do(stack(x, y), s)) \equiv ontable(u, s), \tag{16.9}$$

$$holding(u, do(unstack(x, y), s)) \equiv u = x, \tag{16.10}$$

$$\neg handempty(do(unstack(x, y), s)), \tag{16.11}$$

$$on(u, v, do(unstack(x, y), s)) \equiv on(u, v, s) \wedge \neg(x = u \wedge y = v), \tag{16.12}$$

$$clear(u, do(unstack(x, y), s)) \equiv u = y \vee (clear(u, s) \wedge u \neq x), \tag{16.13}$$

$$ontable(u, do(unstack(x, y), s)) \equiv ontable(u, s), \tag{16.14}$$

$$holding(u, do(pickup(x), s)) \equiv u = x, \tag{16.15}$$

$$\neg handempty(do(pickup(x), s)), \tag{16.16}$$

$$on(u, v, do(pickup(x), s)) \equiv on(u, v, s), \tag{16.17}$$

$$clear(u, do(pickup(x), s)) \equiv clear(u, s) \wedge u \neq x, \tag{16.18}$$

$$ontable(u, do(pickup(x), s)) \equiv ontable(u, s) \wedge x \neq u, \tag{16.19}$$

$$holding(u, do(putdown(x), s)) \equiv holding(u, s) \wedge u \neq x, \tag{16.20}$$

$$handempty(do(putdown(x), s)), \tag{16.21}$$

$$on(u, v, do(putdown(x), s)) \equiv on(u, v, s), \tag{16.22}$$

$$clear(u, do(putdown(x), s)) \equiv u = x \vee clear(u, s), \tag{16.23}$$

$$ontable(u, do(putdown(x), s)) \equiv u = x \vee ontable(u, s). \tag{16.24}$$

Similarly, we can write the following axioms for our second version of the blocks world domain.

$$Poss(move(x, y), s) \equiv x \neq table \wedge clear(x, s) \wedge clear(y, s) \tag{16.25}$$

$$clear(u, do(move(x, y), s)) \equiv$$
$$u = table \vee on(x, u, s) \vee (clear(u, s) \wedge u \neq y), \tag{16.26}$$

$$on(u, v, do(move(x, y), s)) \equiv$$
$$(x = u \wedge y = v) \vee (on(u, v, s) \wedge u \neq x). \tag{16.27}$$

## 16.2 The Frame, the Ramification and the Qualification Problems

The set of axioms (16.1)–(16.24) provides a complete logical characterization of the effects of actions for our first blocks world domain. For each action, it gives necessary and sufficient conditions for it to be executable in any situation, and fully specifies the effects of this action on every fluent. Similarly, the set of axioms (16.25)–(16.27) completely captures the effects of actions for our second blocks world domain.

However, there is something unsatisfying about these two sets of axioms. When we informally described the effects of actions, we did not describe it this way. For

instance, we said that after $stack(x, y)$ is performed, in the resulting new situation, $on(x, y)$ and *handempty* will be true, and $holding(x)$ and $clear(y)$ will no longer be true. We did not have to say, for instance, that if $y$ is initially on the table, it will still be on the table. Many researchers believe that when people remember the effects of an action, they do not explicitly store the facts that are not changed by the action, rather they just remember the changes that this action will bring about. Consequently, when we axiomatize an action, we should only need to specify the changes that will be made by the action. But if we specify in our theory only the changes that an action will make, there is then a problem of how to derive those that are not changed by the action. This problem was identified by McCarthy and Hayes [29] in 1969, and they called it *the frame problem*. For our blocks world example, we can view the frame problem as the problem of looking for an appropriate logic that when given, for example the following so-called "effect axioms" about $stack(x, y)$:

$$on(x, y, do(stack(x, y), s)),$$

$$clear(x, do(stack(x, y), s)),$$

$$\neg clear(y, do(stack(x, y), s)),$$

$$handempty(do(stack(x, y), s)),$$

$$\neg holding(x, do(stack(x, y), s)),$$

will derive a complete specification of the effects of action $stack(x, y)$, like what the set of axioms (16.5)–(16.9) does in first-order logic [21].

The frame problem is one of the most well-known AI problems, if not the most well-known one, and a lot of work has been done on solving it. It motivated much of the early work on nonmonotonic logic (see papers in [6] and Chapter 6). While the problem was identified in the situation calculus, it shows up in other formalisms like the event calculus (Chapter 17), temporal action logics (Chapter 18), and nonmonotonic causal logic (Chapter 19). In fact, the general consensus is that any formalism for reasoning about change will have to deal with it.

McCarthy [27] initially proposed to solve the frame problem by the following generic frame axiom:

$$Holds(p, s) \wedge \neg abnormal(p, a, s) Holds(p, do(a, s)) \tag{16.28}$$

with the abnormality predicate *abnormal* circumscribed. Unfortunately, Hanks and McDermott [10] showed that this approach does not work using by now the infamous Yale Shooting Problem as a counterexample. This is a simple problem with three actions: *wait* (do nothing), *load* (load the gun), and *shoot* (fire the gun). Their effects can be axiomatrized by the following axioms:

$$loaded(do(load, s)), \tag{16.29}$$

$$loaded(s) \supset dead(do(shoot, s)). \tag{16.30}$$

Now suppose $S_0$ is a situation such that the following is true:

$$\neg loaded(S_0) \wedge \neg dead(S_0). \tag{16.31}$$

Hanks and McDermott showed that the circumscription of *abnormal* in the theory {(16.28), (16.29), (16.30), (16.31)} with *Holds* allowed to vary has two models, one in which

$$loaded(do(load, S_0)) \wedge loaded(do(wait, do(load, S_0))) \wedge$$
$$dead(do(shoot, do(wait, do(load, S_0))))$$

is true as desired, and the other in which

$$loaded(do(load, S_0)) \wedge \neg loaded(do(wait, do(load, S_0))) \wedge$$
$$\neg dead(do(shoot, do(wait, do(load, S_0))))$$

is true, which is counter-intuitive as the action *wait*, which is supposed to do nothing, mysteriosly unloaded the gun.

For the next few years, the YSP motivated much of the work on the frame problem, and the frame problem became the focus of the research on nonmonotonic reasoning. In response to the problem, Shoham [37] proposed chronological minimization that prefers changes at later times. Many other proposals were put forward (e.g. [13, 14, 2, 33, 21, 35, 15, 38, 24]).

The thrust of modern solutions to the frame problem is to separate the specification of the effects of actions from the tasks of reasoning about these actions. For instance, given the effect axioms (16.29) and (16.30), one can obtain the following complete specification of the effects of the actions concerned:

$$loaded(do(load, s)),$$
$$dead(do(load, s)) \equiv dead(s),$$
$$loaded(do(shoot, s)) \equiv loaded(s),$$
$$dead(do(shoot, s)) \equiv loaded(s) \vee dead(s),$$
$$loaded(do(wait, s)) \equiv loaded(s),$$
$$dead(do(wait, s)) \equiv dead(s).$$

Now given the initial state axiom (16.31), one can easily infer that $dead(do(shoot, do(wait, do(load, S_0))))$ holds.

This separation between the specification of action theories and the tasks of reasoning under these theories can be done syntactically by distinguishing general effect axioms like (16.29) from specific facts like (16.31) about some particular situations, as in Reiter's solution [33] that we shall describe next. It can also be done by encoding general effect axioms in a special language using predicates like *Caused*, as in Lin's causal theories of action [15] for solving the ramification problem.

### 16.2.1 The Frame Problem—Reiter's Solution

Based on earlier work by Pednault [31], Haas [9] and Schubert [36], Reiter [33, 34] proposed a simple syntactic manipulation much in the style of Clark's predicate completion [4] (see Chapter 7) that turns a set of effect axioms into a set of successor state axioms that completely captures the true value of each fluent in any successor situation. It is best to illustrate Reiter's method by an example. Consider our first blocks world domain, and let us write down all the effect axioms:

$$on(x, y, do(stack(x, y), s)), \tag{16.32}$$
$$clear(x, do(stack(x, y), s)), \tag{16.33}$$
$$\neg clear(y, do(stack(x, y), s)), \tag{16.34}$$
$$handempty(do(stack(x, y), s)), \tag{16.35}$$
$$\neg holding(x, do(stack(x, y), s)), \tag{16.36}$$
$$\neg on(x, y, do(unstack(x, y), s)), \tag{16.37}$$
$$\neg clear(x, do(unstack(x, y), s)), \tag{16.38}$$
$$clear(y, do(unstack(x, y), s)), \tag{16.39}$$
$$\neg handempty(do(unstack(x, y), s)), \tag{16.40}$$
$$holding(x, do(unstack(x, y), s)), \tag{16.41}$$
$$ontable(x, do(putdown(x), s)), \tag{16.42}$$
$$clear(x, do(putdown(x), s)), \tag{16.43}$$
$$handempty(do(putdown(x), s)), \tag{16.44}$$
$$\neg holding(x, do(putdown(x), s)), \tag{16.45}$$
$$\neg ontable(x, do(pickup(x), s)), \tag{16.46}$$
$$\neg clear(x, do(pickup(x), s)), \tag{16.47}$$
$$\neg handempty(do(pickup(x), s)), \tag{16.48}$$
$$holding(x, do(pickup(x), s)). \tag{16.49}$$

Now for each of these effect axioms, transform it into one of the following two forms:

$$\gamma(a, \vec{x}, s) \supset F(\vec{x}, do(a, s)),$$
$$\gamma(a, \vec{x}, s) \supset \neg F(\vec{x}, do(a, s)).$$

For instance, the effect axiom (16.32) can be transformed equivalently into the following axiom:

$$a = stack(x, y) \supset on(x, y, do(a, s)),$$

and the effect axiom (16.34) can be transformed equivalently into the following axiom:

$$(\exists y)a = stack(y, x) \supset \neg clear(x, do(a, s)).$$

Now for each fluent $F$, suppose the following is the list of all such axioms so obtained:

$$\gamma_1^+(a, \vec{x}, s) \supset F(\vec{x}, do(a, s)),$$
$$\ldots$$
$$\gamma_m^+(a, \vec{x}, s) \supset F(\vec{x}, do(a, s)),$$
$$\gamma_1^-(a, \vec{x}, s) \supset \neg F(\vec{x}, do(a, s)),$$
$$\ldots$$
$$\gamma_n^-(a, \vec{x}, s) \supset \neg F(\vec{x}, do(a, s)).$$

Then under what Reiter called *the causal completeness assumption*, which says that the above axioms characterize all the conditions under which action $a$ causes $F$ to become true or false in the successor situation, we conclude the following *successor state axiom* [33] for fluent $F$:

$$F(\vec{x}, do(a, s)) \equiv \gamma^+(a, \vec{x}, s) \vee (F(\vec{x}, s) \wedge \neg\gamma^-(a, \vec{x}, s)), \tag{16.50}$$

where $\gamma^+(a, \vec{x}, s)$ is $\gamma_1^+(a, \vec{x}, s) \vee \cdots \vee \gamma_m^+(a, \vec{x}, s)$, and $\gamma^-(a, \vec{x}, s)$ is $\gamma_1^-(a, \vec{x}, s) \vee \cdots \vee \gamma_n^-(a, \vec{x}, s)$.

For instance, for our first blocks world, we can transform the effect axioms about $clear(x)$ into the following axioms:

$$(\exists y.a = stack(x, y)) \supset clear(x, do(a, s)),$$

$$(\exists y.a = unstack(y, x)) \supset clear(x, do(a, s)),$$

$$a = putdon(x) \supset clear(x, do(a, s)),$$

$$(\exists y.a = stack(y, x)) \supset \neg clear(x, do(a, s)),$$

$$(\exists y.a = unstack(x, y)) \supset \neg clear(x, do(a, s)),$$

$$a = pickup(x) \supset \neg clear(x, do(a, s)).$$

Thus we have the following successor state axiom for $clear(x)$:

$$\begin{aligned} &clear(x, do(a, s)) \equiv \\ &\quad \exists y.a = stack(x, y) \vee \exists y.a = unstack(y, x) \vee \\ &\quad a = putdown(x) \vee clear(x, s) \wedge \\ &\quad \neg[\exists y.a = stack(y, x) \vee \exists y.a = unstack(x, y) \vee a = pickup(x)]. \end{aligned}$$

Once we have a successor state axiom for each fluent in the domain, we will then have an action theory that is complete in the same way as the set of axioms (16.1)–(16.24) is.

This procedure can be given a semantics in nonmonotonic logics, in particular circumscription [27] (see Chapter 6). This in fact has been done by Lin and Reiter [19].

We should also mention that for this approach to work, when generating the successor state axiom (16.50) from effect axioms, one should also assume what Reiter called *the consistency assumption*: the background theory should entail that $\neg(\gamma^+ \wedge \gamma^-)$. Once we have a set of successor state axioms, to reason with them we need the unique names assumption about actions: for each $n$-ary action $A$:

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

and for each distinct actions $A$ and $A'$,

$$A(x_1, \ldots, x_n) \neq A'(y_1, \ldots, y_m).$$

For more details, see [33, 34].

### 16.2.2 The Ramification Problem and Lin's Solution

Recall that the frame problem is about how one can obtain a complete axiomatization of the effects of actions from a set of effect axioms that specifies the changes that the actions have on the world. Thus Reiter's solution to the frame problem makes the assumption that the given effect axioms characterize completely the conditions under which an action can cause a fluent to be true or false. However, in some action domains, providing such a complete list of effect axioms may not be feasible. This is because in these action domains, there are rich domain constraints that can entail new effect axioms. To see how domain constraints can entail new effect axioms, consider again the blocks world. We know that each block can be at only one location: either being held by the robot's hand, on another block, or on the table. Thus when action $stack(x, y)$ causes $x$ to be on $y$, it also makes $holding(x)$ false. The *ramification problem*, first discussed by Finger [5] in 1986, is about how to encode constraints like this in an action domain, and how these constraints can be used to derive the effects of the actions in the domain.

In the situation calculus, for a long time the only way to represent domain constraints was by universal sentences of the form $\forall s.C(s)$. For example, the aforementioned constraint that a block can be (and must be) at only one location in the blocks world can be represented by the following sentences:

$$holding(x, s) \vee ontable(x, s) \vee \exists y.on(x, y),$$
$$holding(x, s) \supset \neg(ontable(x, s) \vee \exists y.on(x, y)),$$
$$ontable(x, s) \supset \neg(holding(x, s) \vee \exists y.on(x, y)),$$
$$(\exists y.on(x, y)) \supset \neg(holding(x, s) \vee ontable(x, s)).$$

So, for example, these axioms and the following effect axiom about $putdown(x)$,

$$ontable(x, do(putdown(x), s))$$

will entail in first-order logic the following effect axiom:

$$\neg holding(x, do(putdown(x), s)).$$

However, domain constraints represented this way may not be strong enough for determining the effects of actions. Consider the suitcase problem from [15]. Imagine a suitcase with two locks and a spring loaded mechanism which will open the suitcase when both of the locks are in the up position. Apparently, because of the spring loaded mechanism, if an action changes the status of the locks, then this action may also cause, as an indirect effect, the suitcase to open.

As with the blocks world, we can represent the constraint that this spring loaded mechanism gives rise to as the following sentence:

$$up(L1, s) \wedge up(L2, s) \supset open(s). \tag{16.51}$$

Although summarizing concisely the relationship among the truth values of the three relevant propositions at any particular instance of time, this constraint is too weak to describe the indirect effects of actions. For instance, suppose that initially the suitcase is closed, the first lock in the down position, and the second lock in the up position.

Suppose an action is then performed to turn up the first lock. Then this constraint is ambiguous about what will happen next. According to it, either the suitcase may spring open or the second lock may get turned down. Although we have the intuition that the former is what will happen, this constraint is not strong enough to enforce that because there is a different mechanism that will yield a logically equivalent constraint. For instance, a mechanism that turns down the second lock when the suitcase is closed and the first lock is up will yield the following logically equivalent one:

$$up(L1, s) \wedge \neg open(s) \supset \neg up(L2, s).$$

So to faithfully represent the ramification of the spring loaded mechanism on the effects of actions, something stronger than the constraint (16.51) is needed. The proposed solution by Lin [15] is to represent this constraint as a causal constraint: (through the spring loaded mechanism) the fact that both of the locks are in the up position *causes* the suitcase to open. To axiomatize this, Lin introduced a ternary predicate $Caused(p, v, s)$, meaning that fluent $p$ is caused to have truth value $v$ in situation $s$. The following are some basic properties of *Caused* [15]:

$$Caused(p, true, s) \supset Holds(p, s), \tag{16.52}$$

$$Caused(p, false, s) \supset \neg Holds(p, s), \tag{16.53}$$

$$true \neq false \wedge (\forall v)(v = true \vee v = false), \tag{16.54}$$

where $v$ is a variable ranging over a new sort *truthValues*.

Let us illustrate how this approach works using the suitcase example. Suppose that $flip(x)$ is an action that flips the status of the lock $x$. Its direct effect can be described by the following axioms:

$$up(x, s) \supset Caused(up(x), false, do(flip(x), s)), \tag{16.55}$$

$$\neg up(x, s) \supset Caused(up(x), true, do(flip(x), s)). \tag{16.56}$$

Assume that $L1$ and $L2$ are the two locks on the suitcase, the spring loaded mechanism is now represented by the following causal rule:

$$up(L1, s) \wedge up(L2, s) \supset Caused(open, true, s). \tag{16.57}$$

Notice that this causal rule, together with the basic axiom (16.52) about causality, entails the state constraint (16.51). Notice also that the physical, spring loaded mechanism behind the causal rule has been abstracted away. For all we care, it may just as well be that the device is not made of spring, but of bombs that will blow open the suitcase each time the two locks are in the up position. It then seems natural to say that the fluent *open* is caused to be true by the fact that the two locks are both in the up position. This is an instance of what has been called *static* causal rules as it mentions only one situation. In comparison, causal statements like the effect axioms (16.55) and (16.56) are *dynamic* as they mention more than one situations.

The above axioms constitute the starting theory for the domain. To describe fully the effects of the actions, suitable frame axioms need to be added. Using predicate *Caused*, a generic frame axiom can be stated as follows [15]: Unless caused otherwise, a fluent's truth value will persist:

$$\neg(\exists v)Caused(p, v, do(a, s)) \supset [Holds(p, do(a, s)) \equiv Holds(p, s)]. \tag{16.58}$$

For this frame axiom to make sense, one needs to minimize the predicate *Caused*. Technically this is done by circumscribing *Caused* in the above set of axioms with all other predicates (*Poss* and *Holds*) fixed. However, given the form of the axioms, this circumscription coincides with Clark's completion of *Caused*, and it yields the following *causation axioms*:

$$\begin{aligned} &Caused(open, v, s) \equiv \\ &\quad v = true \wedge up(L1, s) \wedge up(L2, s), \end{aligned} \tag{16.59}$$

$$\begin{aligned} &Caused(up(x), v, s) \equiv \\ &\quad v = true \wedge (\exists s')[s = do(flip(x), s') \wedge \neg up(x, s')] \vee \\ &\quad v = false \wedge (\exists s')[s = do(flip(x), s') \wedge up(x, s')]. \end{aligned} \tag{16.60}$$

Notice that these axioms entail the two direct effect axioms (16.55), (16.56) and the causal rule (16.57).

Having computed the causal relation, the next step is to use the frame axiom (16.58) to compute the effects of actions. It is easy to see that from the frame axiom (16.58) and the two basic axioms (16.52), (16.53) about causality, one can infer the following pseudo successor state axiom:

$$\begin{aligned} &Holds(p, do(a, s)) \equiv \\ &\quad Caused(p, true, do(a, s)) \vee \\ &\quad Holds(p, s) \wedge \neg Caused(p, false, do(a, s)). \end{aligned} \tag{16.61}$$

From this axiom and the causation axiom (16.60) for the fluent *up*, one then obtains the following real successor state axiom for the fluent *up*:

$$\begin{aligned} &up(x, do(a, s)) \equiv \\ &\quad (a = flip(x) \wedge \neg up(x, s)) \vee (up(x, s) \wedge a \neq flip(x)). \end{aligned}$$

Similarly for the fluent *open*, we have

$$\begin{aligned} &open(do(a, s)) \equiv \\ &\quad [up(L1, do(a, s)) \wedge up(L2, do(a, s))] \vee open(s). \end{aligned}$$

Now from this axiom, first eliminating $up(L1, do(a, s))$ and $up(L2, do(a, s))$ using the successor state axiom for *up*, then using the unique names axioms for actions, and the constraint (16.51) which, as we pointed out earlier, is a consequence of our axioms, we can deduce the following successor state axiom for the fluent *open*:

$$\begin{aligned} open(do(a, s)) \equiv\ & a = flip(L1) \wedge \neg up(L1, s) \wedge up(L2, s) \vee \\ & a = flip(L2) \wedge \neg up(L2, s) \wedge up(L1, s) \vee \\ & open(s). \end{aligned}$$

Obtaining these successor state axioms solves the frame and the ramification problems for the suitcase example.

Lin [15] showed that this procedure can be applied to a general class of action theories, and Lin [18] described an implemented system that can compile these causal

action theories into Reiter's successor state axioms and STRIPS-like systems, and demonstrated the effectiveness of the system by applying it to many benchmark AI planning domains.

### 16.2.3 The Qualification Problem

Finally, we notice that so far we have given the condition for an action $a$ to be executable in a situation $s$, $Poss(a, s)$, directly. It can be argued that this is not a reasonable thing to do. In general, the executability of an action may depend on the circumstances where it is performed. For instance, we have defined $Poss(putdown(x), s) \equiv holding(x, s)$. But if the action is to be performed in a crowd, then we may want to add that for the action to be executable, the robot's hand must not be blocked by someone; and if the robot is running low on battery, then we may want to ensure that the robot is not running out of battery; etc. It is clear that no one can anticipate all these possible circumstances, thus no one can list all possible conditions for an action to be executable ahead of time. This problem of how best to specify the precondition of an action is called the *qualification problem* [26].

One possible solution to this problem is to assume that an action is always executable unless explicitly ruled out by the theory. This can be achieved by maximizing the predicate *Poss*, or in terms of circumscription, circumscribing $\neg Poss$. If the axioms about *Poss* all have the form

$$Poss(A, s) \supset \varphi(s),$$

that is, the user always provides explicit qualifications to an action, then one can compute *Poss* by a procedure like Clark's predicate completion by rewriting the above axiom as:

$$\neg\varphi(s) \supset \neg Poss(A, s).$$

The problem becomes more complex when some domain constraints-like axioms can influence *Poss*. This problem was first recognized by Ginsberg and Smith [7], and discussed in more detailed by Lin and Reiter [19]. For instance, we may want to add into the blocks world a constraint that says "only yellow blocks can be directly on the table". Now if the robot is holding a red block, should she put it down on the table? Probably she should not. This means that this constraint has two roles: it rules out initial states that do not satisfy it, and it forbids agents to perform any action that will result in a successor situation that violates it. What this constraint should not do is to cause additional effects of actions. For instance, it should not be the case that $putdown(x)$ would cause $x$ to become yellow just to maintain this constraint in the successor situation.

Lin and Reiter [19] called those constraints that yield indirect effects of actions *ramification constraints*, and those that yield additional qualifications of actions *qualification constraints*. They are both represented as sentences of the form $\forall s.C(s)$, and it is up to the user to classify which category they belong to.

A uniform way of handling these two kinds of constraints is to use Lin's causal theories of actions as described above. Under this framework, only constraints represented as causal rules using *Caused* can derive new effects of actions, and ordinary situation calculus sentences of the form $\forall s.C(s)$ can only derive new qualifications on

actions. However, for this to work, action effect axioms like (16.55) need to have *Poss* as a precondition:

$$Poss(flip(x), s) \supset [up(x, s) \supset Caused(up(x), false, do(flip(x), s))],$$

and the generic frame axiom (16.58) need to be modified similarly:

$$Poss(a, s) \supset \{\neg(\exists v)Caused(p, v, do(a, s)) \supset [Holds(p, do(a, s)) \equiv Holds(p, s)]\}.$$

In fact, this was how action effect axioms and frame axioms are represented in [15]. An interesting observation made in [15] was that some causal rules may give rise to both new action effects and action qualifications. Our presentation of Lin's causal theories in the previous subsection has dropped *Poss* so that, in line with the presentation in [34], the final successor state axioms do not have *Poss* as a precondition.

## 16.3 Reiter's Foundational Axioms and Basic Action Theories

We have defined the situation calculus as a first-order language with special sorts for situations, actions, and fluents. There are no axioms to constrain these sorts, and all axioms are domain specific given by the user for axiomatizing a particular dynamic system. We have used a binary function $do(a, s)$ to denote the situation resulted from performing $a$ in $s$, thus for a specific finite sequence of actions $a_1, \ldots, a_k$, we have a term denoting the situation resulted from performing the sequence of actions in $s$: $do(a_k, do(a_{k-1}, \ldots, do(a_1, s)\ldots))$. However, there is no way for us to say that one situation is the result of performing *some* finite sequence of actions in another situation. This is needed for many applications, such as planning where the achievability of a goal is defined to be the existence of an executable finite sequence of actions that will make the goal true once executed. We now introduce Reiter's foundational axioms that make this possible.

Briefly, under Reiter's foundational axioms, there is a unique initial situation, and all situations are the result of performing some finite sequences of actions in this initial situation. This initial situation will be denoted by $S_0$, which formally is a constant of sort *situation*.

It is worth mentioning here that while the constant $S_0$ has been used before to informally stand for a starting situation, it was not assumed to be *the* starting situation as under Reiter's situation calculus. It can be said that the difference between Reiter's version of the situation calculus and McCarthy's original version is that Reiter assumed the following foundational axioms that postulate the space of situations as a tree with $S_0$ as the root:

$$do(a, s) = do(a', s') \supset a = a' \wedge s = s', \tag{16.62}$$

$$\forall P.[P(S_0) \wedge \forall a, s.P(s) \supset P(do(a, s))] \supset \forall s\, P(s). \tag{16.63}$$

Axiom (16.63) is a second-order induction axiom that says that for any property $P$, to prove that $\forall s.P(s)$, it is sufficient to show that $P(S_0)$, and inductively, for any situation $s$, if $P(s)$ then for any action $a$, $P(do(a, s))$. In particular, we can conclude that

- $S_0$ is a situation;

- if $s$ is a situation, and $a$ an action, then $do(a, s)$ is a situation;
- nothing else is a situation.

Together with the unique names axiom (16.62), this means that we can view the domain of situations as a tree whose root is $S_0$, and for each action $a$, $do(a, s)$ is a child of $s$. Thus for each situation $s$ there is a unique finite sequence $\alpha$ of actions such that $s = do(\alpha, S_0)$, where for any finite sequence $\alpha'$ of actions, and any situation $s'$, $do(\alpha', s')$ is defined inductively as $do([], s') = s'$, and $do([a|\alpha'], s') = do(a, do(\alpha', s'))$, here we have written a sequence in Prolog notation. Thus there is a one-to-one correspondence between situations and finite sequences of actions under Reiter's foundational axioms, and because of this, Reiter identified situations with finite sequences of actions.

As we mentioned, there is a need to express assertions like "situation $s_1$ is the result of performing a sequence of actions in $s_2$". This is achieved by using a partial order relation $\sqsubset$ on situations: informally $s \sqsubset s'$ if $s'$ is the result of performing a finite nonempty sequence of actions in $s$. Formally, it is defined by the following two axioms:

$$\neg s \sqsubset S_0, \tag{16.64}$$

$$s \sqsubset do(a, s') \equiv s \sqsubseteq s', \tag{16.65}$$

where $s \sqsubseteq s'$ is a shorthand for $s \sqsubset s' \vee s = s'$.

Notice that under the correspondence between situations and finite sequence of actions, the partial order $\sqsubset$ is really the sub-sequence relation: $s \sqsubset s'$ iff the action sequence of $s$ is a sub-sequence of that of $s'$. Thus to say that a goal $g$ is achievable in situation $s$, we write

$$\exists s'.s \sqsubseteq s' \wedge Holds(g, s') \wedge Executable(s, s'),$$

where $Executable(s, s')$ means that the sequence of actions that takes $s$ to $s'$ is executable in $s$, and is defined inductively as:

$$Executable(s, s),$$

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

Reiter's foundational axioms (16.62)–(16.65) make it possible to formally prove many interesting properties such as the achievability of a goal. They also lay the foundation for using the situation calculus to formalize strategic and control information (see, e.g., [16, 17]). They are particularly useful in conjunction with Reiter's successor state axioms, and are part of what Reiter called the *basic action theories* as we proceed to describe now.

To define Reiter's basic action theories, we first need to define the notion of *uniform formulas*. Intuitively, if $\sigma$ is a situation term, then a formula is uniform in $\sigma$ if the truth value of the formula depends only on $\sigma$. Formally, a situation calculus formula is uniform in $\sigma$ if it satisfies the following conditions:

- it does not contain any quantification over situation;
- it does not mention any variables for relational fluents;

- it does not mention any situation term other than $\sigma$;
- it does not mention any predicate that has a situation argument other than *Holds*; and
- it does not mention any function that has a situation argument unless the function is a functional fluent.

Thus $clear(x, s)$ is uniform in $s$ (recall that this is a shorthand for $Holds(clear(x), s)$), but $\forall s.clear(x, s)$ is not as it quantifies over situations. The formula $\forall p.Holds(p, s)$ is not uniform in $s$ either as it contains $p$ which is a variable for relational fluents. Notice that a uniform formula cannot mention domain-independent situation-dependent predicates like *Poss*, $\sqsubset$, and *Caused*. It cannot even contain equalities between situations such as $s = s$, but it can contain equalities between actions and between domain objects such as $x \neq y$, where $x$ and $y$ are variables of sort $block$ in the blocks world.

Another way to view uniform formulas is by using a first-order language without the special *situation* sort. The predicates of this language are relational fluents and other situation independent predicates. The functions are functional fluents, actions, and other situation independent functions. A situation calculus formula $\Phi$ is uniform in $\sigma$ iff there is a formula $\varphi$ in this language such that $\Phi$ is the result of replacing every relational fluent atom $F(t_1, \ldots, t_k)$ in $\varphi$ by $Holds(F(t_1, \ldots, t_k), \sigma)$ (or $F(t_1, \ldots, t_k, \sigma)$) and every functional fluent term $f(t_1, \ldots, t_k)$ by $f(t_1, \ldots, t_k, \sigma)$. In the following, and in Chapter 24 on Cognitive Robotics, this formula $\Phi$ is written as $\varphi[\sigma]$.

Uniform formulas are used in Reiter's action precondition axioms and successor state axioms. In Reiter's basic action theories, an action precondition axiom for an action $A(x_1, \ldots, x_n)$ is a sentence of the form:

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

where $\Pi(x_1, \ldots, x_n, s)$ is a formula uniform in $s$ and whose free variables are among $x_1, \ldots, x_n, s$. Thus whether $A(x_1, \ldots, x_n)$ can be performed in a situation $s$ depends entirely on $s$.

We have seen successor state axioms for relational fluents (16.50). In general, in Reiter's basic action theories, a successor state axiom for an $n$-ary relational fluent $F$ is a sentence of the form

$$F(x_1, \ldots, x_n, do(a, s)) \equiv \Phi_F(x_1, \ldots, x_n, a, s), \tag{16.66}$$

where $\Phi_F$ is a formula uniform in $s$, and whose free variables are among $x_1, \ldots, x_n, a, s$.

Similarly, if $f$ is an $(n + 1)$-ary functional fluent, then a successor state axiom for it is a sentence of the form

$$f(x_1, \ldots, x_n, do(a, s)) = v \equiv \varphi(x_1, \ldots, x_n, v, a, s), \tag{16.67}$$

where $\varphi$ is a formula uniform in $s$, and whose free variables are among $x_1, \ldots, x_n, v, a, s$.

Notice that requiring the formulas $\Phi_F$ and $\varphi$ in successor state axioms to be uniform amounts to making Markov assumption in systems and control theory: the effect of an action depends only on the current situation.

We can now define Reiter's basic action theories [34]. A basic action theory $\mathcal{D}$ is a set of axioms of the following form:

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

where

- $\Sigma$ is the set of the four foundational axioms (16.62)–(16.65).
- $\mathcal{D}_{ss}$ is a set of successor state axioms. It must satisfy the following *functional fluent consistency property*: if (16.67) is in $\mathcal{D}_{ss}$, then

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

- $\mathcal{D}_{ap}$ is a set of action precondition axioms.
- $\mathcal{D}_{una}$ is the set of unique names axioms about actions.
- $\mathcal{D}_{S_0}$ is a set of sentences that are uniform in $S_0$. This is the knowledge base for the initial situation $S_0$.

The following theorem is proved by Pirri and Reiter [32].

**Theorem 16.1** (Relative satisfiability). *A basic action theory $\mathcal{D}$ is satisfiable iff $\mathcal{D}_{una} \cup \mathcal{D}_{S_0}$ is.*

As we mentioned above, the basic action theories are the starting point to solve various problems in dynamic systems. Many of these problems can be solved using basic action theories by first-order deduction. But some of them require induction. Those that require induction are typically about proving general assertions of the form $\forall s.C(s)$, such as proving that a certain goal is not achievable. For instance, consider the basic action theory $\mathcal{D}$ where $\mathcal{D}_{ap} = \emptyset$, $\mathcal{D}_{ss} = \{\forall a, s.loaded(do(a, s)) \equiv loaded(s)\}$, and $\mathcal{D}_{S_0} = \{loaded(S_0)\}$. It is certainly true that $\mathcal{D} \models \forall s.loaded(s)$. But proving this formally requires induction.

The ones that can be done in first-order logic include checking whether a sequence of ground actions is executable in $S_0$ and the temporal projection problem, which asks whether a formula holds after a sequence of actions is performed in $S_0$. One very effective tool for solving these problems is *regression*,[2] which transforms a formula

$$\varphi(do([\alpha_1, \ldots, \alpha_n], S_0))$$

that is uniform in $do([\alpha_1, \ldots, \alpha_n], S_0)$ to a formula $\varphi'(S_0)$ that is uniform in $S_0$ such that

$$\mathcal{D} \models \varphi(do([\alpha_1, \ldots, \alpha_n], S_0)) \quad \text{iff} \quad \mathcal{D}_{una} \cup \mathcal{D}_{S_0} \models \varphi'(S_0).$$

If $\varphi(do([\alpha_1, \ldots, \alpha_n], S_0))$ does not have functional fluents, then the regression can be defined inductively as follows: the regression of $\varphi(S_0)$ is $\varphi(S_0)$, and inductively, if

[2] Reiter [34] defined regression for a more general class of formulas that can contain *Poss* atoms.

$\alpha$ is an action term, and $\sigma$ a situation term, then the regression of $\varphi(do(\alpha, \sigma))$ is the regression of the formula obtained by replacing in $\varphi(do(\alpha, \sigma))$ each relational fluent atom $F(\vec{t}, do(\alpha, \sigma))$ by $\Phi_F(\vec{t}, \alpha, \sigma)$, where $\Phi_F$ is the formula in the right side of the successor state axiom (16.66) for $F$. When $\varphi$ contains functional fluents, the definition of regression is more involved, see [34].

For instance, given the following successor state axioms:

$$F(do(a, s)) \equiv a = A \vee F(s),$$
$$G(do(a, s)) \equiv (a = B \wedge F(s)) \vee G(s),$$

the regression of $G(do(B, do(A, S_0)))$ is the regression of

$$(B = B \wedge F(do(A, S_0))) \vee G(do(A, S_0)),$$

which is the following sentence about $S_0$:

$$(B = B \wedge (A = A \vee F(S_0))) \vee (A = B \wedge F(S_0)) \vee G(S_0),$$

which is equivalent to *true*.

Using regression, we can check the executability of a sequence of actions in $S_0$, say $[stack(A, B), pickup(C), putdown(C)]$, as follows:

1. This sequence of actions is executable in $S_0$ iff the following formulas are entailed by $\mathcal{D}$:

$$Poss(stack(A, B), S_0),$$
$$Poss(pickup(C), do(stack(A, B), S_0)),$$
$$Poss(putdown(C), do(pickup(C), do(stack(A, B), S_0))).$$

2. Use action precondition axioms to rewrite the above sentences into uniform sentences. For instance, the first two sentences can be rewritten into the following sentences:

$$clear(B, S_0) \wedge holding(A, S_0),$$
$$handempty(do(stack(A, B), S_0)) \wedge ontable(C, (do(stack(A, B), S_0))) \wedge$$
$$clear(C, (do(stack(A, B), S_0))).$$

3. Regress the uniform sentences obtained in step 2, and check whether the regressed formulas are entailed by $\mathcal{D}_{una} \cup \mathcal{D}_{S_0}$.

## 16.4 Applications

The situation calculus provides a rich framework for solving problems in dynamic systems. Indeed, Reiter [34] showed that many such problems can be formulated in the situation calculus and solved using a formal situation calculus specification. He even had the slogan "No implementation without a SitCalc specification".

The first application of the situation calculus is in planning where an agent needs to figure out a course of actions that will achieve a given goal. Green [8] formulated

this problem as a theorem proving task in the situation calculus:

$$T \models \exists s.G(s),$$

where $T$ is the situation calculus theory for the planning problem, and $G$ the goal. Green's idea was that if one can find a proof of the above theorem constructively, then a plan can be read off from the witness situation term in the proof. He actually implemented a planning system based on this idea using a first-order theorem prover. For various reasons, the system could solve only extremely simple problems. Some researchers believe that Green's idea is correct. What one needs is a good way to encode domain specific control knowledge as the situation calculus sentences to direct the theorem prover intelligently.

One reason that Green's system performed poorly was that the theory $T$ that encodes the planning problem is not very effective. Assuming that the planning problem is specified by a basic action theory, Reiter implemented a planner in Prolog that can efficiently make use of domain-specific control information like that in [1]. It can even do open-world planning where the initial situation is not completely specified. For more details see [34].

The situation calculus has also been used to formalize and reason about computer programs. Burstall used it to formalize Algol-like programs [3]. Manna and Waldinger used it to formalize general assignments in Algol 68 [22], and later Lisp imperative programs [23].

More recently, Lin and Reiter used it to axiomatize logic programs with negation-as-failure [20]. The basic idea is as follows. A rule (clause) $P \leftarrow G$ means that whenever $G$ is proved, we can use this rule to prove $P$. Thus we can name this rule by an action so that the consequence of the rule becomes the effect of the action, and the body of the rule becomes the context under which the action will have the effect.[3] Formally, for each rule

$$F(t_1, \ldots, t_n) \leftarrow Q_1, \ldots, Q_k, \text{not } Q_{k+1}, \ldots, \text{not } Q_m$$

Lin and Reiter introduced a corresponding $n$-ary action $A$, and axiomatized it with the following axioms:

$$\begin{aligned}
&Poss(A(\vec{x}), s),\\
&[\exists \vec{y}_1 Holds(Q_1, s) \wedge \cdots \wedge \exists \vec{y}_k Holds(Q_k, s) \wedge\\
&\quad \neg(\exists \vec{y_{k+1}}, s) Holds(Q_{k+1}, s) \wedge \cdots \wedge\\
&\quad \neg(\exists \vec{y_m}, s) Holds(Q_m, s)] \supset F(t_1, \ldots, t_n, do(A(\vec{x}), s)),
\end{aligned}$$

where $\vec{y}_i$ is the tuple of variables in $Q_i$ but not in $F(t_1, \ldots, t_n)$.

Notice that $\neg(\exists \vec{y}_i, s) Holds(Q_i, s)$ means that the goal $Q_i$ is not achievable (provable) no matter how one instantiate the variables that are in $Q_i$ but not in the head of the rule. This is meant to capture the "negation-as-failure" feature of the operator "not" in logic programming.

Now for a logic program $\Pi$, which is a finite set of rules, one can apply Reiter's solution to the effect axioms obtained this way for all rules in $\Pi$, and obtain a set of

[3] Alternatively, one could also view the body of a rule as the precondition of the corresponding action.

successor state axioms, one for each predicate occurring in the program.[4] Thus for each program $\Pi$, we have a corresponding basic action theory[5] $\mathcal{D}$ for it with

$$\mathcal{D}_{S_0} = \{F(\vec{x}, S_0) \equiv \mathit{false} \mid F \text{ is a predicate in } \Pi\}.$$

In other words, in the initial situation, all fluents are false. Now query answering in a logic program becomes planning under the corresponding basic action theory.

As it turned out, this formalization of logic programs in the situation calculus yields a semantics that is equivalent to Gelfond and Lifschitz's stable model semantics. This situation calculus semantics can be used to formally study some interesting properties of logic programs. For instance, it can be proved that program unfolding preserves this semantics. More interestingly, under this semantics and Reiter's foundational axioms, derivations under a program are isomorphic to situations. Thus those operators that constrain derivations in logic programming can be axiomatized in the situations calculus by their corresponding constraints on situations. Based on this idea, Lin proposed a situation calculus semantics for the "cut" operator in logic programming [16].

The most significant application so far is the use of the situation calculus as a working language for Cognitive Robotics. For details about this application, the reader is referred to the chapter on Cognitive Robotics in this Handbook.

## 16.5 Concluding Remarks

What we have described so far is just the elemental of the situation calculus. The only thing that we care about an action so far is its logical effects on the physical environment. We have ignored many other aspects of actions, such as their durations and their effects on the agent's mental state. We have also assumed that actions are performed sequentially one at a time and that they are the only force that may change the state of the world. These and other issues in reasoning about action have been addressed in the situation calculus, primarily as a result of using the situation calculus as the working language for Cognitive Robotics. We refer the interested readers to Chapter 24 and [34].

## Acknowledgements

I would like to thank Gerhard Lakemeyer, Hector Levesque, and Abhaya Nayak for their very useful comments on an earlier version of this article.

## Bibliography

[1] F. Bacchus and F. Kabanza. Using temporal logics to express search control knowledge for planning. *Artificial Intelligence*, 16:123–191, 2000.

[4] If a predicate does not occur as the head of a rule, then add an axiom to say that this relation does not hold for any situation.

[5] Strictly speaking, it is not a basic action theory as the right side of a successor state axiom may contain $\exists s.Q(s)$ when there is a negation in front of $Q$ in a rule, thus is not a uniform formula.

[2] A.B. Baker. Nonmonotonic reasoning in the framework of the situation calculus. *Artificial Intelligence*, 49:5–23, 1991.
[3] R.M. Burstall. Formal description of program structure and semantics in first-order logic. In B. Meltzer and D. Michie, editors. *Machine Intelligence*, vol. 5, pages 79–98. Edinburgh University Press, Edinburgh, 1969.
[4] K.L. Clark. Negation as failure. In H. Gallaire and J. Minker, editors. *Logics and Databases*, pages 293–322. Plenum Press, New York, 1978.
[5] J. Finger. Exploiting constraints in design synthesis. PhD thesis, Department of Computer Science, Stanford University Stanford, CA, 1986.
[6] M.L. Ginsberg. *Readings in Nonmonotonic Reasoning*. Morgan Kaufmann, San Mateo, CA, 1987.
[7] M.L. Ginsberg and D.E. Smith. Reasoning about action II: The qualification problem. *Artificial Intelligence*, 35:311–342, 1988.
[8] C.C. Green. Application of theorem proving to problem solving. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-69)*, pages 219–239, 1969.
[9] 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 on Reasoning about Action*, pages 343–348. Morgan Kaufmann Publishers, Inc, San Jose, CA, 1987.
[10] S. Hanks and D. McDermott. Nonmonotonic logic and temporal projection. *Artificial Intelligence*, 33:379–412, 1987.
[11] D. Harel. *First-Order Dynamic Logic*. *Lecture Notes in Computer Science*, vol. 68. Springer-Verlag, New York, 1979.
[12] H. Levesque, R. Reiter, Y. Lespérance, F. Lin, and R. Scherl. GOLOG: A logic programming language for dynamic domains. *Journal of Logic Programming*, 31:59–84, 1997 (Special issue on Reasoning about Action and Change).
[13] V. Lifschitz. Pointwise circumscription. In *Proceedings of the Fifth National Conference on Artificial Intelligence (AAAI-86)*, pages 406–410. Philadelphia, PA, 1986.
[14] V. Lifschitz. Formal theories of action. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence (IJCAI-87)*, pages 966–972, 1987.
[15] F. Lin. Embracing causality in specifying the indirect effects of actions. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95)*, pages 1985–1993. IJCAI Inc., Morgan Kaufmann, San Mateo, CA, 1995.
[16] F. Lin. Applications of the situation calculus to formalizing control and strategic information: The Prolog cut operator. *Artificial Intelligence*, 103:273–294, 1998.
[17] F. Lin. Search algorithms in the situation calculus. In H. Levesque and F. Pirri, editors. *Logical Foundations for Cognitive Agents: Contributions in Honor of Ray Reiter*, pages 213–233. Springer, Berlin, 1999.
[18] F. Lin. Compiling causal theories to successor state axioms and STRIPS-like systems. *Journal of Artificial Intelligence Research*, 19:279–314, 2003.
[19] F. Lin and R. Reiter. State constraints revisited. *Journal of Logic and Computation*, 4(5):655–678, 1994 (Special Issue on Actions and Processes).
[20] F. Lin and R. Reiter. Rules as actions: A situation calculus semantics for logic programs. *J. of Logic Programming*, 31(1–3):299–330, 1997.

[21] F. Lin and Y. Shoham. Provably correct theories of action. *Journal of the ACM*, 42(2):293–320, 1995.
[22] Z. Manna and R. Waldinger. Problematic features of programming languages: A situational-calculus approach. *Acta Informatica*, 16:371–426, 1981.
[23] Z. Manna and R. Waldinger. The deductive synthesis of imperative LISP programs. In *Proceedings of the Sixth National Conference on Artificial Intelligence (AAAI-87)*, pages 155–160, Seattle, WA, 1987.
[24] N. McCain and H. Turner. Causal theories of action and change. In *Proceedings of the 14th National Conference on Artificial Intelligence (AAAI-97)*, pages 460–465. Menlo Park, CA, 1997, AAAI Press.
[25] J. McCarthy. Situations, actions and causal laws. In M. Minsky, editor. *Semantic Information Processing*, pages 410–417. MIT Press, Cambridge, MA, 1968.
[26] J. McCarthy. Epistemological problems of Artificial Intelligence. In *IJCAI-77*, pages 1038–1044, Cambridge, MA, 1977.
[27] J. McCarthy. Applications of circumscription to formalizing commonsense knowledge. *Artificial Intelligence*, 28:89–118, 1986.
[28] J. McCarthy. Actions and other events in situation calculus. In *Proceedings of the Eighth International Conference on Principles of Knowledge Representation and Reasoning (KR2002)*, pages 615–628, 2002.
[29] J. McCarthy and P. Hayes. Some philosophical problems from the standpoint of artificial intelligence. In B. Meltzer and D. Michie, editors. *Machine Intelligence*, vol. 4, pages 463–502. Edinburgh University Press, Edinburgh, 1969.
[30] N.J. Nilsson. *Principles of Artificial Intelligence*. Morgan Kaufmann, Los Altos, CA, 1980.
[31] E.P. Pednault. ADL: Exploring the middle ground between STRIPS and the situation calculus. In *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning (KR'89)*, pages 324–332. Morgan Kaufmann Publishers, Inc, 1989.
[32] F. Pirri and R. Reiter. Some contributions to the metatheory of the situation calculus. *J. ACM*, 46(3):325–361, 1999.
[33] R. Reiter. The frame problem in the situation calculus: a simple solution (sometimes) and a completeness result for goal regression. In V. Lifschitz, editor. *Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy*, pages 418–420. Academic Press, San Diego, CA, 1991.
[34] R. Reiter. *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*. The MIT Press, 2001.
[35] E. Sandewall. *Features and Fluents. A Systematic Approach to the Representation of Knowledge about Dynamical Systems*, vol. I. Oxford University Press, 1994.
[36] L.K. Schubert. Monotonic solution to the frame problem in the situation calculus: an efficient method for worlds with fully specified actions. In H. Kyberg, R. Loui, and G. Carlson, editors. *Knowledge Representation and Defeasible Reasoning*, pages 23–67. Kluwer Academic Press, Boston, MA, 1990.
[37] Y. Shoham. Chronological ignorance: experiments in nonmonotonic temporal reasoning. *Artificial Intelligence*, 36:279–331, 1988.
[38] M. Thielscher. Ramification and causality. *Artificial Intelligence*, 89:317–364, 1997.

This page intentionally left blank