Rules as Actions: A Situation Calculus Semantics for Logic Programs

Fangzhen Lin and Ray Reiter

▷ We propose a novel semantics for logic programs with negation by viewing the application of a clause in a derivation as an action in the situation calculus. Program clauses are then identified with situation calculus effect axioms as they are understood in axiomatic theories of actions. We then solve the frame problem for these effect axioms using a recent approach of Reiter [21], and identify the resulting collection of axioms with the semantics of the original logic program. An interesting consequence of this approach is that the logic programming negation-as-failure operator inherits its nonmonotonicity from the nonmonotonicity associated with the frame problem.

One advantage of our proposal is that like Clark's completion semantics, ours is also formulated explicitly in classical logic. To illustrate the usefulness of our semantics, we prove sufficient conditions for two logic programs to be equivalent, and use this to verify the correctness of the well-known unfolding program transformation operator. We also discuss applications of this framework to formalizing search control operators in logic programming. ◁

Keywords. Semantics of logic programs. Reasoning about actions. The situation calculus. The frame problem.

1. Introduction

In this paper we propose a novel semantics for logic programs in the situation calculus. One of the advantages of our proposal is that like Clark's completion semantics, it is explicitly formulated in classical logic. For this reason, it is suitable for proving properties of logic programs such as the correctness of various program transformation operators.

Fellow of the Canadian Institute for Advanced Research

Address correspondence to Department of Computer Science, University of Toronto, Toronto, Canada M5S 1A4. Email: {fl,reiter}@ai.toronto.edu. http://www.cs.toronto.edu/˜cogrobo/

655 Avenue of the Americas, New York, NY 10010 0743-1066/94/$7.00

The basic idea of our proposal is very simple. We consider the application of a clause in a derivation to be an action in the situation calculus (McCarthy [15]). Executing a clause makes the head of the clause true in the new situation whenever the body of the clause is true in the current situation. Program clauses are then identified with situation calculus effect axioms as they are understood in axiomatic theories of actions. We then solve the frame problem for these effect axioms using a recent approach of Reiter [21], and identify the resulting collection of axioms with the semantics of the original logic program.

This paper is organized as follows. In the next section, we briefly review the situation calculus and the frame problem. Section 3 provides the necessary logical preliminaries, and Section 4 defines our situation calculus semantics for logic programs. Section 5 shows some relationships between our semantics and Wallace's [31], and also relates our semantics to the stable model semantics [3]. Section 6 formulates conditions for two logic programs to be equivalent and applies this result to verifying the correctness of the unfolding program transformation operator. Section 7 discusses other potential applications of our semantics, while Section 8 provides some concluding remarks.

2. An Informal Introduction to the Situation Calculus[1]

2.1. Intuitive Ontology for the Situation Calculus

The situation calculus (McCarthy [15]) is a first order language (with, as we shall see later, some second order features) specifically designed for representing dynamically changing worlds. All changes to the world are the result of named *actions*. A possible world history, which is simply a sequence of actions, is represented by a first order term called a *situation*. The constant S_0 is used to denote the *initial situation*, namely that situation in which no actions have yet occurred. There is a distinguished binary function symbol do; $do(\alpha, s)$ denotes the successor situation to s resulting from performing the action α. Actions may be parameterized. For example, $put(x, y)$ might stand for the action of putting object x on object y, in which case $do(put(A, B), s)$ denotes the situation resulting from placing A on B when the world is in situation s. Notice that in the situation calculus, actions are denoted by first order terms, and situations (world histories) are also first order terms. For example, $do(putdown(A), do(walk(L), do(pickup(A), S_0)))$ is a situation denoting the world history consisting of the sequence of actions [pickup(A), walk(L), putdown(A)]. Notice that the sequence of actions in a history, in the order in which they occur, is obtained from a situation term by reading off the actions from right to left.

Generally, the values of relations in a dynamic world will vary from one situation to the next. Such relations are called *fluents*, and are denoted by predicate symbols taking a situation term as one of their arguments. The convention we shall adopt is that the situation argument of a fluent will always be its last argument. For example, in a mobile robot environment, we might have a relational fluent $closeTo(r, x, s)$ meaning that in situation s the robot r is close to the object x.

[1] This section is included in order to make this paper as self contained as possible. With minor differences, it is the same as that of Levesque et al [7].

2.2. Axiomatizing Actions and their Effects in the Situation Calculus

Actions have *preconditions* – necessary and sufficient conditions that characterize when the action is physically possible. For example, in a blocks world, we might have:[2]

$$Poss(pickup(x), s) \equiv \\ [(\forall z)\neg holding(z, s)] \wedge nexto(x, s) \wedge \neg heavy(x).$$

World dynamics are specified by *effect axioms*. These describe the effects of a given action on the fluents – the causal laws of the domain. For example, a robot dropping a fragile object causes it to be broken:

$$Poss(drop(r, x), s) \wedge fragile(x, s) \supset broken(x, do(drop(r, x), s)). \tag{2.1}$$

Exploding a bomb next to an object causes it to be broken:

$$Poss(explode(b), s) \wedge nexto(b, x, s) \supset broken(x, do(explode(b), s)). \tag{2.2}$$

A robot repairing an object causes it to be not broken:

$$Poss(repair(r, x), s) \supset \neg broken(x, do(repair(r, x), s)). \tag{2.3}$$

2.3. The Frame Problem

As first observed by McCarthy and Hayes [17], axiomatizing a dynamic world requires more than just action precondition and effect axioms. So-called *frame axioms* are also necessary. These specify the action *invariants* of the domain, namely, those fluents which remain unaffected by a given action. For example, a robot dropping things does not affect an object's color:

$$Poss(drop(r, x), s) \wedge color(y, c, s) \supset color(y, c, do(drop(r, x), s)).$$

A frame axiom describing how the fluent *broken* remains unaffected:

$$Poss(drop(r, x), s) \wedge \neg broken(y, s) \wedge [y \neq x \vee \neg fragile(y, s)] \\ \supset \neg broken(y, do(drop(r, x), s)).$$

The problem introduced by the need for such frame axioms is that we can expect a vast number of them. Only relatively few actions will affect the truth value of a given fluent; all other actions leave the fluent invariant. For example, an object's color is not changed by picking things up, opening a door, going for a walk, electing a new prime minister of Canada, etc. This is problematic for the axiomatizer – she must think of all these axioms – and it is problematic for the theorem proving system – it must reason efficiently in the presence of so many frame axioms.

2.3.1. What Counts as a Solution to the Frame Problem?

Suppose the person responsible for axiomatizing an application domain has specified all of the causal laws for the world being axiomatized. More precisely, she has succeeded in writing down *all* the effect axioms, i.e. for each fluent F and each action A which can cause F's truth value to change, axioms of the form

$$Poss(A, s) \wedge R(\vec{x}, s) \supset (\neg)F(\vec{x}, do(A, s)).$$

[2] In formulas, free variables are considered to be universally quantified from the outside. This convention will be followed throughout the paper.

Here, R is a first order formula specifying the contextual conditions under which the action A will have its specified effect on F.

A solution to the frame problem is a systematic procedure for generating, from these effect axioms, all the frame axioms. If possible, we also want a *parsimonious* representation for these frame axioms (because in their simplest form, there are too many of them).

2.4. A Simple Solution to the Frame Problem

By appealing to earlier ideas of Haas [5], Schubert [24] and Pednault [19], Reiter [21] proposes a simple solution to the frame problem, which we illustrate with an example. Suppose that (2.1), (2.2), and (2.3) are all the effect axioms for the fluent *broken*, i.e. they describe all the ways that an action can change the truth value of *broken*. We can rewrite (2.1) and (2.2) in the logically equivalent form:

$$\begin{aligned}&Poss(a,s) \wedge [(\exists r)\{a = drop(r,x) \wedge fragile(x,s)\}\\ &\qquad \vee (\exists b)\{a = explode(b) \wedge nexto(b,x,s)\}]\\ &\qquad\qquad \supset broken(x, do(a,s)).\end{aligned} \tag{2.4}$$

Similarly, consider the negative effect axiom (2.3) for *broken*; this can be rewritten as:

$$Poss(a,s) \wedge (\exists r) a = repair(r,x) \supset \neg broken(x, do(a,s)). \tag{2.5}$$

In general, we can assume that the effect axioms for a fluent F have been written in the forms:

$$Poss(a,s) \ \wedge\ \gamma_F^+(\vec{x},a,s) \ \supset\ F(\vec{x}, do(a,s)), \tag{2.6}$$

$$Poss(a,s) \ \wedge\ \gamma_F^-(\vec{x},a,s) \ \supset\ \neg F(\vec{x}, do(a,s)). \tag{2.7}$$

Here $\gamma_F^+(\vec{x},a,s)$ is a formula describing under what conditions doing the action a in situation s leads the fluent F to become true in the successor situation $do(a,s)$; similarly $\gamma_F^-(\vec{x},a,s)$ describes the conditions under which performing a in s results in F becoming false in the next situation. The solution to the frame problem of [21] rests on a *completeness assumption*, which is that the causal axioms (2.6) and (2.7) characterize all the conditions under which action a can lead to a fluent $F(\vec{x})$ becoming true (respectively, false) in the successor situation. In other words, axioms (2.6) and (2.7) describe all the causal laws affecting the truth values of the fluent F. Therefore, if action a is possible and $F(\vec{x})$'s truth value changes from *false* to *true* as a result of doing a, then $\gamma_F^+(\vec{x},a,s)$ must be *true* and similarly for a change from *true* to *false*. Reiter [21] shows how to derive a *successor state axiom* of the following form from the causal axioms (2.6) and (2.7) and the completeness assumption.

Successor State Axiom

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

This single axiom embodies a solution to the frame problem. Notice that this axiom universally quantifies over actions a. In fact, this is one way in which a parsimonious solution to the frame problem is obtained.

Applying this to our example about breaking things, we obtain the following successor state axiom:

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

It is important to note that the above solution to the frame problem presupposes that there are no *state constraints*, as for example in the blocks world constraint: $(\forall s).on(x, y, s) \supset \neg on(y, x, s)$. Such constraints sometimes implicitly contain effect axioms (so-called indirect effects), in which case the above completeness assumption will not be true.

In what follows, we shall provide a semantics for logic programs, with negation, by treating the application of a rule (clause) in a derivation as an action in the situation calculus. Program clauses will then be identified with effect axioms. By solving the frame problem exactly as just described, we shall obtain a situation calculus representation of the program which will serve as the program's logical semantics.

As is well known, solutions to the frame problem are nonmonotonic, in the sense that the above completeness assumption (the given effect axioms are all and only the effect axioms) is a kind of closed world assumption. The addition of a new effect axiom to an earlier axiomatization for some domain may invalidate any solution to the frame problem obtained with the earlier axioms. This intuition has led to a large body of research on nonmonotonic solutions to the frame problem (e.g. [16, 26, 8, 9, 13]). In view of our situation calculus semantics for logic programming, it will follow that *negation-as-failure inherits its nonmonotonicity from the nonmonotonicity associated with the frame problem.*

3. Logical Preliminaries

3.1. The Language of the Situation Calculus

The language $\mathcal{L}$ of the situation calculus is many-sorted, second-order, with equality. We assume the following sorts: *situation* for situations, *action* for actions, and *object* for everything else. We also assume the following domain independent predicates and functions:

- A constant S_0 of sort *situation* denoting the initial situation.
- A binary function do - $do(a, s)$ denotes the situation resulting from performing action a in situation s.
- A binary predicate $Poss$ - $Poss(a, s)$ means that action a is possible (executable) in situation s. In this paper we shall assume that actions are always executable, i.e. $(\forall a, s)Poss(a, s)$. So technically, there is no real need for this predicate in this paper. We keep it, however, in order to be consistent with the general framework of (Reiter [21] and Lin and Reiter [11]).
- A binary predicate $<$ over situations. We shall follow convention, and write $<$ in infix form. By $s < s'$ we mean that s' can be obtained from s by a sequence of executable actions. As usual, $s \leq s'$ will be a shorthand for $s < s' \vee s = s'$.

We assume a finite number of *fluents*, which are predicate symbols of arity $object^n \times situation$, $n \geq 0$, and are domain dependent. We also assume a finite number of function symbols of arity $object^n \rightarrow action$, $n \geq 0$, for actions, and a finite number of function symbols of arity $object^n \rightarrow object$, $n \geq 0$.

3.2. Axiomatizing the Situation Calculus

We shall need the following foundational axioms (Lin and Reiter [11]) for the situation calculus:

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

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

$$(\forall P)[P(S_0) \wedge (\forall a, s)(P(s) \supset P(do(a, s))) \supset (\forall s)P(s)],$$

$$\neg s < S_0,$$

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

Intuitively, the first two axioms are unique names assumptions. They eliminate cycles, and merging. The third axiom is second order induction. It amounts to the domain closure axiom that every situation is obtained from the initial one by repeatedly applying the function *do*.[3]As we shall see, induction will play an important role in this paper. The last two axioms define $<$ inductively.

Notice the similarity between these axioms and the Peano foundational axioms for number theory. However, unlike Peano arithmetic which has a unique successor function, we have a class of successor functions here represented by the function *do*. In the following, we shall denote by Σ the set of the above axioms.

3.3. Logic Programs

An *atom* p is an expression of the form $F(t_1, ..., t_n)$, where F is a fluent of arity $object^n \times situation$, and $t_1, \ldots, t_n$ are terms of sort $object$. Notice that an atom is not a formula in the situation calculus. It is an expression obtained from an atomic situation calculus formula by suppressing its situation argument.

A *literal* is either an atom, or an expression of the form `not` p, where p is an atom. In addition, an equality formula of the form $t = t'$ is a *literal*, where t and t' are terms of sort $object$. Again, notice that, except for equality literals, literals are not formulas in the language of the situation calculus.

A *goal* G is an expression of the form

$$l_1 \;\texttt{\&}\; \cdots \;\texttt{\&}\; l_n$$

where $n \geq 0$, and $l_1, \ldots, l_n$ are literals. A *clause* is an expression of the form

$$F(\vec{x}) \;\texttt{:-}\; G$$

where F is a fluent symbol, $\vec{x}$ is a tuple of distinct variables of length n, $n \geq 0$, and G is a goal. Notice that according to this definition, the head of a clause must not mention constants and compound terms. This, however, does not restrict the

[3]For a detailed discussion of the use of induction in the situation calculus, see (Reiter [22]).

generality of our notion of clauses. For any terms $t_1, ..., t_n$ of sort *object*, we can take an expression of the form

$$F(t_1, ..., t_n) \ \texttt{:-}\ G$$

to be a *shorthand* for the following clause:

$$F(\vec{x}) \ \texttt{:-}\ \vec{x} = \vec{t} \ \texttt{\&}\ G$$

where $\vec{x} = (x_1, ..., x_n)$ is a tuple of fresh variables not mentioned in G or in $\vec{t}$. Generally, for any vectors $\vec{t} = (t_1, ..., t_k)$ and $\vec{t'} = (t'_1, ..., t'_k)$ of terms of the same length, if $\vec{t} = \vec{t'}$ appears in a goal, then it stands for

$t_1 = t'_1 \ \& \ \cdots \ \& \ t_k = t'_k$,

and if $\vec{t} = \vec{t'}$ appears in a situation calculus formula, then it stands for

$t_1 = t'_1 \wedge \cdots \wedge t_k = t'_k$.

Finally, a *normal program* is a finite set of clauses. In the following, normal programs will simply be called programs. The *definition* of a fluent symbol F in a program P is the set of clauses in P that mention F in their head.

Since we will be interpreting clauses as formulas of the situation calculus, we need a way to interpret literals in the situation calculus. Given a literal l, and a situation term st, we define $l[st]$ as follows:

1. If l is an atom of the form $F(t_1, ..., t_n)$, then $l[st]$ is $F(t_1, ..., t_n, st)$, i.e. it is the formula obtained from l by putting st back as its last argument.
2. If l is a negated atom of the form `not` $F(t_1, ..., t_n)$, then $l[st]$ is the formula $\neg(\exists s)F(t_1, ..., t_n, s)$. Notice that in this case, the truth value of the formula $l[st]$ is independent of the situation st. *This is our interpretation of the negation-as-failure operator "*`not`*" in the situation calculus.*
3. If l is an equality formula of the form $t = t'$, then $l[st]$ is l.

Now if G is a goal of the form $l_1 \& \cdots \& l_n$, and st a situation term, then we define $G[st]$ to be the formula

$l_1[st] \wedge \cdots \wedge l_n[st]$.

3.4. Clark's Completion

Since we shall often refer to Clark's completion [1] in this paper, we briefly review it here.

A clause of the form

$$F(\vec{x}) \ \texttt{:-}\ l_1 \ \texttt{\&} \ \cdots \ \texttt{\&}\ l_n$$

stands for the following implication about F:

$$(\exists \vec{y})(l'_1 \wedge \cdots \wedge l'_n) \supset F(\vec{x}), \tag{3.1}$$

where $\vec{y}$ is the tuple of variables that appear in some l_i, $1 \leq i \leq n$, but are distinct from variables in $\vec{x}$, and if l is an atom then l' is l, and if l is `not` p, then l' is $\neg p$.

Given a logic program P, if

$$F(\vec{x}) \ \text{:-}\ G_1$$
$$\vdots$$
$$F(\vec{x}) \ \text{:-}\ G_m$$

is the definition of F, then the Clark completion of F in P is the following sentence:

$$F(\vec{x}) \equiv [(\exists \vec{y_1})G'_1 \wedge \cdots \wedge (\exists \vec{y_m})G'_m],$$

where $(\exists \vec{y_i})G'_i$, $1 \leq i \leq m$, is as the left hand side of the implication (3.1). Notice that if $m = 0$, i.e. there are no clauses in P about F, then the Clark completion of F is $F(\vec{x}) \equiv false$.

The Clark completion of a program P is then the set consisting of the following axioms:

1. For each predicate F in P, the Clark completion of F in P.
2. Unique names axioms for the function symbols appearing in P.

Clark's completion is perhaps the simplest semantics for logic programs. It replaces rules in a logic program by logical axioms in first-order logic. The main problem with it is that it is too weak for logic programs with cycles and recursion (see, for example, [14]).

Our proposed semantics will be very much in the same style as Clark's completion, but it will also handle cycles and recursion correctly.

4. A Situation Calculus Semantics for Logic Programs

On our intuition about logic programs, clauses are treated as rules, so that the application of such a rule in the process of obtaining a derivation is like performing an action. So a clause of the form

$$F(\vec{x}) \ \text{:-}\ G$$

is like the specification of the effects of an action; if G holds currently, then $F(\vec{x})$ will hold after the action is performed. Taking this intuition seriously, suppose that we name this clause by the action $A(\vec{x})$ in our situation calculus language. Then we have the following axiom (an *effect axiom*) describing the effect of A:

$$Poss(A(\vec{x}), s) \supset (G[s] \supset F(\vec{x}, do(A(\vec{x}), s)).$$

Recall from section 3.1 that we have assumed that actions are always possible:

$$Poss(a, s) \equiv true.$$

Thus the above effect axiom is equivalent to

$$G[s] \supset F(\vec{x}, do(A(\vec{x}), s)).$$

Let $\vec{y}$ be the tuple of variables in G which are not in $\vec{x}$, then we can rewrite the above axiom as

$$(\exists \vec{y})G[s] \wedge a = A(\vec{x}) \supset F(\vec{x}, do(a, s)). \qquad (4.1)$$

Notice the similarity, but not the formal identity, between this transformation and that leading up to the formation of the Clark completion of a predicate.

Example 4.1. Suppose that $gf(x, y)$ is the action naming the following clause:

```
grandfather(x,y) :- parent(x,z) & parent(z,y) & not female(x)
```

Then we have the following effect axiom:

$$(\exists z)[parent(x,z,s) \wedge parent(z,y,s)] \wedge \neg(\exists s')female(x,s') \wedge a = gf(x,y) \supset \\ grandfather(x,y,do(a,s)).$$

Now suppose that P is a program and F a fluent. Suppose the following are the corresponding effect axioms of the form (4.1) for the clauses in the definition of F in P:

$$(\exists \vec{y_1})G_1[s] \wedge a = A_1(\vec{x}) \supset F(\vec{x}, do(a,s)),$$

$$\vdots$$

$$(\exists \vec{y_n})G_n[s] \wedge a = A_n(\vec{x}) \supset F(\vec{x}, do(a,s)).$$

Then, by solving the frame problem for fluent F as described in Section 2.4, we obtain the following *successor state axiom* for F:

$$F(\vec{x}, do(a,s)) \equiv \{(\exists \vec{y_1})G_1[s] \wedge a = A_1(\vec{x}) \vee \cdots \vee \\ (\exists \vec{y_n})G_n[s] \wedge a = A_n(\vec{x}) \vee \\ F(\vec{x}, s)\}. \tag{4.2}$$

Intuitively, the successor state axiom for F says that the fluent is true in a successor situation iff either it is true in the current situation, or the action names one of the clauses in the definition of F and the body of that clause is true in the current situation. In particular, if the definition of F in the program P is empty, then (4.2) becomes

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

In the following, we call (4.2) the *successor state axiom for F with respect to P*. Notice the similarity between this axiom and the Clark completion of F.

We can now define the "meaning" of logic programs in the situation calculus. We assume that for each clause there is a unique action symbol that names the clause, and has the same number of arguments as that of the predicate in the head of the clause.

Definition 4.1. Let P be a program. The *action theory $\mathcal{D}$ for P* is

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

where

- Σ is the set of foundational axioms given in Section 3.2.
- $\mathcal{D}_{ss}$ is the set of successor state axioms for the fluents with respect to P.
- $\mathcal{D}_{una}$ is the set consisting of the following *unique names axioms*:

  $$f(\vec{x}) \neq g(\vec{y}) \tag{4.3}$$

  for every pair f, g of distinct function symbols, and

  $$f(\vec{x}) = f(\vec{y}) \supset \vec{x} = \vec{y} \tag{4.4}$$

  for every function symbol f. Notice that constants are considered to be 0-ary functions. We remark that for the function do, (4.4) is the same as one of our foundational axioms in Σ. We also remark that $\mathcal{D}_{una}$ includes unique names axioms for the actions introduced to name the program clauses.

- $\mathcal{D}_{S_0}$ is:

  $\{F(\vec{x}, S_0) \equiv false \mid F \text{ is a fluent}\}.$

  In other words, in the initial situation, all fluents are false.

Notice that only the set $\mathcal{D}_{ss}$ of successor state axioms is dependent on the clauses in P. All other sets in the above definition either are domain independent or depend only on the vocabulary of P.

Proposition 4.1. Suppose, as in (4.2), that fluent F's successor state axiom has the form

$$F(\vec{x}, do(a, s)) \equiv \phi(\vec{x}, a, s) \vee F(\vec{x}, s),$$

where $\phi(\vec{x}, a, s)$ is any first order formula whose free variables are among $\vec{x}, a, s$. Suppose further that $F(\vec{x}, S_0) \equiv false$. Then the foundational and unique names axioms for the situation calculus, together with these two sentences entail:

$$F(\vec{x}, s) \equiv (\exists a', s')[do(a', s') \leq s \wedge \phi(\vec{x}, a', s')].$$

PROOF. The proof is by induction. The case $s = S_0$ is immediate. So, assume the result for situation s. We must prove

$$F(\vec{x}, do(a, s)) \equiv (\exists a', s')[do(a', s') \leq do(a, s) \wedge \phi(\vec{x}, a', s')].$$

$\Longrightarrow$
Assume $F(\vec{x}, do(a, s))$. We must prove

$$(\exists a', s')[do(a', s') \leq do(a, s) \wedge \phi(\vec{x}, a', s')]. \tag{4.5}$$

By F's successor state axiom, we have $\phi(\vec{x}, a, s) \vee F(\vec{x}, s)$.
Case 1: $\phi(\vec{x}, a, s)$. Then take $a' = a$ and $s' = s$ in (4.5) and we are done.
Case 2: $F(\vec{x}, s)$. By induction hypothesis, we know that for some α and σ, $do(\alpha, \sigma) \leq s$ and $\phi(\vec{x}, \alpha, \sigma)$. So take $a' = \alpha$ and $s' = \sigma$ in (4.5), leaving us to prove $do(\alpha, \sigma) \leq do(a, s)$; this follows immediately from $do(\alpha, \sigma) \leq s$ and the foundational axioms for the situation calculus.

$\Longleftarrow$
Assume, for some α and σ that $do(\alpha, \sigma) \leq do(a, s)$ and that $\phi(\vec{x}, \alpha, \sigma)$. We must prove $F(\vec{x}, do(a, s))$, or equivalently, by F's successor state axiom, that $\phi(\vec{x}, a, s) \vee F(\vec{x}, s)$.
Case 1: $do(\alpha, \sigma) = do(a, s)$. Then by the unique names axiom for situations, $\alpha = a$ and $\sigma = s$, and we are done.
Case 2: $do(\alpha, \sigma) < do(a, s)$. Then by the foundational axioms, $do(\alpha, \sigma) \leq s$, so by the induction hypothesis, we have $F(\vec{x}, s)$.

Corollary 4.1. Let P be a program, and $\mathcal{D}$ its action theory. Then, for any fluent F,

$$\mathcal{D} \models (\forall \vec{x})(\forall s, s').[F(\vec{x}, s) \wedge s \leq s'] \supset F(\vec{x}, s').$$

This informs us that if a fluent ever becomes true, it will never again become false.

Corollary 4.2. Let P be a program, $\mathcal{D}$ its action theory, and F a fluent. Suppose the successor state axiom for F in $\mathcal{D}$ is of the form (4.2). Then $\mathcal{D}$ entails the following closed form solution *for F:*

$$F(\vec{x}, s) \equiv \{(\exists s')(do(A_1(\vec{x}), s') \leq s \wedge (\exists \vec{y_1})G_1[s']) \vee \cdots \vee \\ (\exists s')(do(A_n(\vec{x}), s') \leq s \wedge (\exists \vec{y_n})G_n[s'])\}. \quad (4.6)$$

Intuitively, this closed form solution (4.6) for F says that F holds in some situation s iff there is an earlier situation s' in which an action occurs that causes F to be true.

Definition 4.2. Let P be a program, and G a goal. A substitution σ, not necessarily ground, is an *answer* for G iff

$\mathcal{D} \models (\forall \vec{x})(\exists s) G\sigma[s]$,

where $G\sigma$ is the result of simultaneously substituting for variables in G according to σ, and $\vec{x}$ are all the free variables mentioned in $G\sigma$.

Therefore query answering in logic programs literally becomes planning in the style of (Green [4]) in the situation calculus.

As we can see from this definition of an answer, we are primarily interested in consequences of the form $(\exists s)G[s]$. One nice property about these consequences of action theories is that they commute over conjunctions:

Proposition 4.2. Let P be a program, and $\mathcal{D}$ its action theory. For any goals G_1 and G_2, whose free variables together are $\vec{x}$,

$$\mathcal{D} \models (\forall \vec{x})\{(\exists s)(G_1 \,\&\, G_2)[s] \equiv (\exists s)G_1[s] \wedge (\exists s)G_2[s]\}.$$

Proof. First, define a function f as follows:

$$f(s, S_0) = s,$$

$$f(s, do(a, s')) = do(a, f(s, s')).$$

Intuitively, $f(s, s')$ is that situation reached by performing those actions which took you from S_0 to s, followed by those actions which took you from S_0 to s'. Next, we prove two lemmas:

Lemma 4.1. For every fluent F,

$$\mathcal{D} \models (\forall \vec{x}, s, s').F(\vec{x}, s) \supset F(\vec{x}, f(s, s')).$$

Proof of Lemma:
The proof is by induction on s', using the induction axiom in the foundational axioms of the situation calculus. The case $s' = S_0$ is trivial. Assume the induction hypothesis for s'; we prove, for each fluent F, that

$$(\forall \vec{x}, s).F(\vec{x}, s) \supset (\forall a)F(\vec{x}, f(s, do(a, s'))).$$

By the definition of f, this is equivalent to proving

$$(\forall \vec{x}, s).F(\vec{x}, s) \supset (\forall a)F(\vec{x}, do(a, f(s, s'))).$$

By (4.2), F's successor state axiom has the form $F(\vec{x}, do(a,s)) \equiv F(\vec{x}, s) \vee \phi(\vec{x}, a, s)$. Using this and the induction hypothesis, the result follows.

Lemma 4.2. For every fluent F,

$$\mathcal{D} \models (\forall \vec{x}, s, s').F(\vec{x}, s) \supset F(\vec{x}, f(s', s)).$$

Proof of Lemma:
The proof is by induction on s. As induction hypothesis we take:

$$(\forall s). \bigwedge [(\forall \vec{x}, s').F(\vec{x}, s) \supset F(\vec{x}, f(s', s))],$$

where the conjunction is over the finitely many fluents F of our situation calculus language. When $s = S_0$, we must prove: $(\forall \vec{x}, s').F(\vec{x}, S_0) \supset F(\vec{x}, s')$. This follows from Corollary 4.1. Assume the induction hypothesis for s; we prove, for each fluent F, that

$$(\forall \vec{x}, a, s').F(\vec{x}, do(a,s)) \supset F(\vec{x}, f(s', do(a,s))).$$

By the definition of f, this is equivalent to proving

$$(\forall \vec{x}, a, s').F(\vec{x}, do(a,s)) \supset F(\vec{x}, do(a, f(s', s))).$$

By F's successor state axiom (4.2), $F(\vec{x}, do(a,s)) \equiv F(\vec{x}, s) \vee \phi(\vec{x}, a, s)$, where ϕ is a disjunction of formulas of the form $(\exists \vec{y})G[s] \wedge a = A(\vec{x})$. So we must prove

$$(\forall \vec{x}, a, s').F(\vec{x}, s) \vee \phi(\vec{x}, a, s) \supset F(\vec{x}, f(s', s)) \vee \phi(\vec{x}, a, f(s', s)).$$

By the induction hypothesis, this simplifies to proving

$$(\forall \vec{x}, a, s').\phi(\vec{x}, a, s) \supset F(\vec{x}, f(s', s)) \vee \phi(\vec{x}, a, f(s', s)).$$

Now $\phi(\vec{x}, a, s)$ is a disjunction of formulas of the form $(\exists \vec{y})G[s] \wedge a = A(\vec{x})$, where G is a goal. Hence, by the induction hypothesis,

$$(\forall \vec{x}, a, s').\phi(\vec{x}, a, s) \supset \phi(\vec{x}, a, f(s', s)).$$

This completes the proof of the lemma.

Now, to prove the proposition, notice first that by the properties of first order logic, the following is valid:

$$(\forall \vec{x})\{(\exists s)(G_1 \,\&\, G_2)[s] \supset (\exists s)G_1[s] \wedge (\exists s)G_2[s]\}.$$

To prove that

$$\mathcal{D} \models (\forall \vec{x})\{(\exists s)G_1[s] \wedge (\exists s)G_2[s] \supset (\exists s)(G_1 \& G_2)[s]\},$$

it is sufficient, with no loss of generality, to show that for any two fluents F and F',

$$\mathcal{D} \models (\forall \vec{x}, \vec{y})\{(\exists s)F(\vec{x}, s) \wedge (\exists s)F'(\vec{y}, s) \supset (\exists s)[F(\vec{x}, s) \wedge F'(\vec{y}, s)]\}.$$

To prove this, assume $(\exists s)F(\vec{X}, s)$ and $(\exists s)F'(\vec{Y}, s)$ for vectors $\vec{X}$ and $\vec{Y}$ of Skolem constants. Then for constants σ and σ', we have $F(\vec{X}, \sigma)$ and $F'(\vec{Y}, \sigma')$. We must prove

$$(\exists s)[F(\vec{X}, s) \wedge F'(\vec{Y}, s)]. \tag{4.7}$$

By Lemma 1, we have $F(\vec{X}, f(\sigma, \sigma'))$. By Lemma 2, we have $F'(\vec{Y}, f(\sigma, \sigma'))$. By taking $s = f(\sigma, \sigma')$ we have proved (4.7).

By Corollary 4.2 and Proposition 4.2, we see that for every program P, the action theory for P entails the Clark completion of P with every atom $F(\vec{t})$ replaced by $(\exists s)F(\vec{t}, s)$:

Theorem 4.1. Let P be a program, $\mathcal{D}$ its action theory, and F a fluent. Suppose the successor state axiom for F in $\mathcal{D}$ is (4.2), and G_i is l_{i1} & $\cdots$ & l_{ik_i} for $1 \le i \le n$. Then $\mathcal{D}$ entails the Clark completion for F:

$$\begin{aligned}(\exists s)F(\vec{x}, s) \equiv \{&(\exists \vec{y_1})\{(\exists s)l_{11}[s] \wedge \cdots \wedge (\exists s)l_{1k_1}[s]\} \vee \cdots \vee \\ &(\exists \vec{y_n})\{(\exists s)l_{n1}[s] \wedge \cdots \wedge (\exists s)l_{nk_n}[s]\}\}.\end{aligned}$$

Example 4.2. Consider the logic program P_1 with the single rule

```
F :- not F
```

By Theorem 4.1, P_1's action theory entails

$$(\exists s)F(s) \equiv \neg(\exists s)F(s),$$

an inconsistent sentence. Thus the basic action theory for P_1 is inconsistent.

Consider the logic program P_2 with the following two rules:

```
F :- not Q
Q :- not F
```

Theorem 4.1 yields the following entailment of P_2's action theory:

$$(\exists s)F(s) \equiv \neg(\exists s)Q(s).$$

Thus we can distinguish two classes of models of the action theory for P_2, one in which $\neg(\exists s)Q(s)$ holds so the first rule is applicable but not the second, and the other in which $\neg(\exists s)F(s)$ holds so the second rule is applicable, but not the first.

Now consider the following logic program:[4]

```
F :- not Q
Q :- not F
R :- F
R :- Q
```

We prove that R is an answer to this program, which is to say, that $(\exists s)R(s)$ is an entailment of our situation calculus semantics. By Theorem 4.1, this program's action theory entails the following Clark completion:

$$(\exists s)F(s) \equiv \neg(\exists s')Q(s'),$$

$$(\exists s)Q(s) \equiv \neg(\exists s')F(s'),$$

$$(\exists s)R(s) \equiv (\exists s)F(s) \vee (\exists s)Q(s).$$

These first order sentences entail $(\exists s)R(s)$, so that R is an answer to this program.

[4] Thanks to Vladimir Lifschitz for suggesting this example.

When there is recursion, our action theory may be stronger than Clark's completion, as the following example shows.

Example 4.3. Consider the definite program P_1 with the following clauses:

```
ancestor(x,y) :- parent(x,y)
ancestor(x,y) :- ancestor(x,z) & ancestor(z,y)
parent(x,y) :- x=John & y=Joe
parent(x,y) :- x=Joe & y=Bill
parent(x,y) :- x=Joe & y=Susan
```

Let $A_1(x,y), A_2(x,y), B_1(x,y), B_2(x,y), B_3(x,y)$ be the actions naming these five clauses, respectively. For the fluent *ancestor*, we have the following two effect axioms:

$$parent(x,y,s) \land a = A_1(x,y) \supset ancestor(x,y,do(a,s)),$$
$$(\exists z)[ancestor(x,z,s) \land ancestor(z,y,s)] \land a = A_2(x,y) \supset \\ \quad ancestor(x,y,do(a,s)).$$

Thus we have the following successor state axiom for *ancestor*:

$$ancestor(x,y,do(a,s)) \equiv \{a = A_1(x,y) \land parent(x,y,s) \lor \\ \quad a = A_2(x,y) \land (\exists z).ancestor(x,z,s) \land ancestor(z,y,s) \lor \\ \quad ancestor(x,y,s)\}.$$

Similarly, we obtain the following successor state axiom for *parent*:

$$parent(x,y,do(a,s)) \equiv \{x = John \land y = Joe \land a = B_1(x,y) \lor \\ \quad x = Joe \land y = Bill \land a = B_2(x,y) \lor \\ \quad x = Joe \land y = Susan \land a = B_3(x,y) \lor \\ \quad parent(x,y,s)\}.$$

Let $\mathcal{D}_1$ be the action theory for P_1. By Theorem 4.1, we have:

$$\mathcal{D}_1 \models (\forall x,y)\{(\exists s)parent(x,y,s) \equiv [(x = John \land y = Joe) \lor \\ \quad (x = Joe \land y = Bill) \lor (x = Joe \land y = Susan)]\}.$$

For *ancestor*, Theorem 4.1 yields

$$\mathcal{D}_1 \models (\forall x,y)\{(\exists s)ancestor(x,y,s) \equiv [(\exists s)parent(x,y,s) \lor \\ \quad (\exists z).(\exists s)ancestor(x,z,s) \land (\exists s)ancestor(z,y,s)]\}.$$

This is too weak to give a solution for *ancestor* (because of the recursion). However, by the successor state axioms in $\mathcal{D}_1$, using induction on situations (Section 2.2), we can show that

$$\mathcal{D}_1 \models (\forall x,y)\{(\exists s)ancestor(x,y,s) \equiv [(x = John \land y = Joe) \lor \\ \quad (x = Joe \land y = Bill) \lor (x = Joe \land y = Susan) \lor \\ \quad (x = John \land y = Bill) \lor (x = John \land y = Susan)]\}.$$

This shows that our semantics is strictly stronger than Clark's completion.

Now consider the program P_2 which is P_1 together with the following clauses:

```
childless(x) :- not haschild(x)
haschild(x) :- parent(x,y)
```

Let $C(x)$ and $D(x)$ be the corresponding two actions. The successor state axioms for *parent* and *ancestor* with respect to P_2 are the same as those with respect to P_1. The successor state axioms for *childless* and *haschild* are:

$$childless(x, do(a, s)) \equiv [a = C(x) \wedge \neg(\exists s')haschild(x, s') \vee \\ childless(x, s)],$$

$$haschild(x, do(a, s)) \equiv [a = D(x) \wedge (\exists y)parent(x, y, s) \vee \\ haschild(x, s)].$$

Let $\mathcal{D}_2$ be the action theory for P_2. By Theorem 4.1, we have

$$\begin{aligned}\mathcal{D}_2 \models (\forall x).(\exists s)childless(x, s) &\equiv \neg(\exists s)haschild(x, s) \\ &\equiv \neg(\exists s)(\exists y)parent(x, y, s) \\ &\equiv \neg(x = John \vee x = Joe).\end{aligned}$$

In the next section, we shall show that our action theory semantics for logic programs is closely related to a recent semantics proposed by Wallace [31], and is essentially the same as the stable model semantics [3] when we consider only Herbrand models.

5. Wallace's Semantics

Wallace's basic idea [31] can be summarized as follows: Given a logic program P, first obtain from P another program P', then consider the semantics of P to be the Clark completion of P'. Wallace proposes several ways for obtaining the new program P' from P. We shall consider the one that is most closely related to our semantics, and that will in turn relate our semantics to the stable model semantics of [3].

The following definition is adapted from [31]. Let P be a logic program. The *tightened* program P' of P contains precisely the following clauses:

1. For each clause

   $$F(\vec{x}) \texttt{ :- } l_1 \texttt{ \& } \cdots \texttt{ \& } l_n$$

   in P, P' contains the clause

   $$F(\vec{x}, s(n)) \texttt{ :- } l'_1 \texttt{ \& } \cdots \texttt{ \& } l'_k$$

   where $s(n)$ denotes the successor of the natural number n, and if $l_i = G(\vec{t})$ is an atom, then l'_i is $G(\vec{t}, n)$; if l_i is a negative atom, then $l'_i = l_i$.
2. For each predicate $F(\vec{x})$ in P, P' contains the clause:

   $$F(\vec{x}) \texttt{ :- } F(\vec{x}, n)$$

For example, consider the following program adapted from [31]:

$$\begin{aligned}F(x) &\texttt{ :- } Q(x) \texttt{ \& not } R(x) \\ Q(a) &\texttt{ :- } \\ R(x) &\texttt{ :- } R(x)\end{aligned}$$

The tightened version of this program is:

$$
\begin{array}{l}
F(x, s(n)) \;\texttt{:-}\; Q(x, n) \;\texttt{\& not}\; R(x) \\
Q(a, s(n)) \;\texttt{:-} \\
R(x, s(n)) \;\texttt{:-}\; R(x, n) \\
F(x) \;\texttt{:-}\; F(x, n) \\
Q(x) \;\texttt{:-}\; Q(x, n) \\
R(x) \;\texttt{:-}\; R(x, n)
\end{array}
$$

Notice that the Clark completion of the tightened program yields, for example, the following completion axiom for $F(x)$:

$$(\forall x)(F(x) \equiv (\exists n)F(x, n)),$$

and the following completion axiom for $F(x, n)$:

$$(\forall x, n)(F(x, n) \equiv (\exists n')(n = s(n') \wedge Q(x, n')) \wedge \neg R(x)).$$

Notice the similarity between this axiom and our successor state axiom for the fluent F, in particular, when $R(x)$ in the above axiom is replaced by $(\exists n')R(x, n')$ according to the completion of $R(x)$. The differences are that instead of situations, Wallace uses natural numbers, and instead of actions and the function *do*, Wallace uses the successor function.

We assume that 0 is a constant symbol denoting the number zero. So the Herbrand models of the Clark completion of tightened programs contains precisely the following terms about numbers:

$$0, s(0), s(s(0)), \ldots$$

Theorem 5.1. Let P be a logic program, $\mathcal{D}$ its action theory, and $\mathcal{C}$ the Clark completion of the tightened version of P. Then,

1. *If M is a Herbrand model of $\mathcal{C}$, then there is a Herbrand model M' of $\mathcal{D}$ such that for any predicate F in P, and any tuple of Herbrand terms $\vec{t}$,*

   $$M \models (\exists n)F(\vec{t}, n) \Leftrightarrow M' \models (\exists s)F(\vec{t}, s).$$

2. *If M is a Herbrand model of $\mathcal{D}$, then there is a Herbrand model M' of $\mathcal{C}$ such that for any predicate F in P, and any tuple of Herbrand terms $\vec{t}$,*

   $$M \models (\exists s)F(\vec{t}, s) \Leftrightarrow M' \models (\exists n)F(\vec{t}, n).$$

Proof. First, notice that the Herbrand domain for situations is

$$\{S_0, do(\alpha_1, S_0), do(\alpha_2, S_0), \ldots, do(\alpha_1, do(\alpha_1, S_0)), do(\alpha_2, do(\alpha_1, S_0)), \ldots\}.$$

So any Herbrand interpretation will satisfy the foundational axioms Σ of the situation calculus (Section 3.2).

Let M be a Herbrand model of $\mathcal{C}$. Construct a Herbrand interpretation M' with respect to $\mathcal{D}$ as follows. For any fluent F, let

1. $M' \models (\forall \vec{x})\neg F(\vec{x}, S_0)$.
2. Inductively, for any situation term $do(\alpha, S)$, and any tuple of Herbrand terms $\vec{t}$, if α does not name any clause with F as its head in P, then

   $$M' \models F(\vec{t}, do(\alpha, S)) \Leftrightarrow M' \models F(\vec{t}, S),$$

   and if α names a clause with F as its head, say

   $$F(\vec{x}) \;\texttt{:-}\; F_1(\vec{t_1}) \;\texttt{\& not}\; F_2(\vec{t_2})$$

in P, then $M' \models F(\vec{t}, do(\alpha, S))$ iff for some tuple $\vec{u}$ of Herbrand terms of the same length as $\vec{y}$, the tuple of variables in the above clause but not in $\vec{x}$, $M' \models F_1(\vec{t_1}, S)(\vec{x}, \vec{y}/\vec{t}, \vec{u})$ and $M \models \neg(\exists n) F_2(\vec{t_2}, n)(\vec{x}, \vec{y}/\vec{t}, \vec{u})$, where $F_1(\vec{t_1}, S)(\vec{x}, \vec{y}/\vec{t}, \vec{u})$ is obtained from $F_1(\vec{t_1}, S)$ by replacing x_i in $\vec{x} = (x_1, \ldots, x_n)$ by the corresponding term t_i in $\vec{t}$, and y_i in $\vec{y}$ by the corresponding term in $\vec{u}$. Similarly for $F_2(\vec{t_2}, n)(\vec{x}, \vec{y}/\vec{t}, \vec{u})$.

It remains to show that

1. M' is a model of $\mathcal{D}$.
2. For any fluent F, and any tuples of Herbrand terms $\vec{t}$,

   $M \models (\exists n) F(\vec{t}, n) \Leftrightarrow M' \models (\exists s) F(\vec{t}, s)$.

Notice that by our construction of M', (1) follows straightforwardly from (2). To prove the "$\Rightarrow$" part of (2), suppose that for some natural number N, $M \models F(\vec{t}, N)$. We show by induction on N that there is a situation S such that $M' \models F(\vec{t}, S)$. The case for $N = 0$ is vacuous because $M \models \neg F(\vec{t}, 0)$. Inductively, suppose this is true for any predicate F', and any $N < K$. Suppose now $M \models F(\vec{t}, K)$. Then since M is a model of the Clark completion of the tightened version of P, there must be a clause with F as its head, say

$$F(\vec{x}) \texttt{ :- } F_1(\vec{t_1}) \texttt{ \& not } F_2(\vec{t_2})$$

in P such that for some tuple $\vec{u}$ of Herbrand terms of the same length as $\vec{y}$, the tuple of variables in the above clause, but different from those in $\vec{x}$,

$$M \models F_1(\vec{t_1}, K-1)(\vec{x}, \vec{y}/\vec{t}, \vec{u}) \wedge \neg(\exists n) F_2(\vec{t_2}, n)(\vec{x}, \vec{y}/\vec{t}, \vec{u}).$$

By the inductive assumption, there is a situation S_1 such that $M' \models F_1(\vec{t_1}, S_1)(\vec{x}, \vec{y}/\vec{t}, \vec{u})$. Now let $A(\vec{x})$ be the action naming the above clause for F. By the construction of M', we have that $M' \models F(\vec{t}, do(A(\vec{t}), S_1))$. This completes the inductive step, thus the "$\Rightarrow$" part of (2). The "$\Leftarrow$" part of (2) can be proved similarly by doing induction on situations.

This completes the proof for the first half of the theorem. The proof of the second half is similar.

¿From this theorem and Theorem 8 in [31] that relates Wallace's semantics to the stable model semantics, we have:

Corollary 5.1. Let P be a program, and $\mathcal{D}$ its action theory. A set $\mathcal{S}$ of ground atoms is a stable model of P iff there is a Herbrand model M of $\mathcal{D}$ such that for any ground atom $F(\vec{t})$, $F(\vec{t}) \in \mathcal{S}$ iff $M \models (\exists s) F(\vec{t}, s)$.

For any logic program P, Wallace also defines the *full completion* of P to be the Clark completion of the tightened version of P together with appropriate induction axioms for natural numbers, and shows that for any ground atom p, p is entailed by the full completion iff it is in the *success set* of the tight tree semantics of P as defined in (van Gelder [28]), and $\neg p$ is entailed by the full completion iff p is in the *finite failure set* of P. Since our foundational axioms in Σ already include an induction axiom, this result carries over to our semantics as well.

Wallace [31] also relates his semantics to some other well-known ones such as (Fitting [2], Kunen [6], Przymusinski [20], and van Gelder and Ross and Schlipf [29]). Many of the results there can be inherited here. Wallace also argues the advantages of having a semantics in first-order logic. The same arguments apply to our semantics as well.

Admittedly, compared to Wallace's elegant approach, ours seems complicated. However, there are some important reasons for appealing to actions and their axiomatization within the situation calculus.

1. Appealing to theories of actions as they are normally understood in artificial intelligence reveals the connection between the classical frame problem and the semantics for negation-as-failure.
2. By treating rule applications as first-order objects, we can formally reason about them within the situation calculus. This becomes important when we come to formalize search control operators in logic programming. Because actions and situations are first order terms, and because a situation denotes the sequence of actions (history) that have occurred thus far, a situation in our logic programming semantics is a record of *all the derivations that have been performed thus far, in the order in which they have been performed.* To date, most formal analyses of logic programming have ignored their "dirty aspects" like the cut operator. In essence, these operators place certain constraints on reachable situations, i.e. on the permitted derivation histories. These conditions are normally rather complicated and require the ability to talk formally about derivation histories, which our situation calculus-based semantics does provide. We shall say more about this issue in Section 7 below.

Technically, this paper also goes beyond that of (Wallace [31]) in defining an equivalence relation on logic programs, and proving conditions for two logic programs to be equivalent. This is the goal of the next section.

6. Program Transformations

One reason for a formal semantics of a programming language is to study sound program transformation techniques. To this end, we first need a notion of equivalence between two logic programs.

6.1. An Equivalence Relation

Let P and P' be two programs, and $\mathcal{D}$ and $\mathcal{D}'$ their respective action theories. Normally $\mathcal{D}$ and $\mathcal{D}'$ will not be compatible. For example, any action in P but not in P' will have no effect according to $\mathcal{D}'$. Given our definition of answers to queries, it is natural then to say that P and P' are equivalent iff they give the same answer to every query, i.e., for any goal G, $\mathcal{D} \models (\exists s)G[s]$ iff $\mathcal{D}' \models (\exists s)G[s]$. However, this definition does not seem to be fine-grained enough. For example, the following program

```
F :- not Q
Q :- not F
R :- F
```

gives the same answer to every query as the following one:

```
F :- not Q
Q :- not F
R :- Q
```

But intuitively, we don't want them to be equivalent because there is a model of the action theory of the first program in which

$$\neg(\exists s)Q(s) \wedge (\exists s)F(s) \wedge (\exists s)R(s)$$

holds, but there is no model of the action theory of the second program that satisfies this sentence.[5] This suggests that we should define program equivalence model-theoretically.

Let P be a logic program, $\mathcal{D}$ its action theory. We call a theory T an *answer theory* of P iff for every structure M, M is a model of T iff there is a model M' of $\mathcal{D}$ such that M and M' agree on $(\exists s)G[s]$, for any goal G. In the following, we write this agreement relation as $M \sim M'$. Formally, $M \sim M'$ iff

1. M and M' have the same domain for sort *object*. Recall that this is the sort for entities other than actions and situations.
2. For any goal G, and any variable assignment σ,[6]

$$M, \sigma \models (\exists s)G[s] \text{ iff } M', \sigma \models (\exists s)G[s].$$

It is clear that if both T and T' are answer theories of P, then T and T' are logically equivalent. So if T is an answer theory of P, then we can say that T is *the* answer theory.

Notice that by Proposition 4.2, condition 2 holds for arbitrary $(\exists s)G[s]$ iff it holds for any $(\exists s)F(\vec{x}, s)$, where F is a fluent:

Proposition 6.1. $M \sim M'$ iff

1. *M and M' have the same domain for sort* object.
2. *For any fluent F, and any variable assignment σ,*

$$M, \sigma \models (\exists s)F(\vec{x}, s) \textit{ iff } M', \sigma \models (\exists s)F(\vec{x}, s).$$

It turns out that the answer theory of P can be considered to be the result of *remembering only* $(\exists s)F(\vec{x}, s)$, for every fluent F in $\mathcal{D}$ (Lin and Reiter [12]). Moreover, according to the results in (Lin and Reiter [12]), the answer theory of P always exists, and can be expressed as a finite second-order theory, but that in general, no first-order answer theory need exist.

We now have the following definition:[7]

Definition 6.1. Two logic programs P and P' are *equivalent* iff their answer theories are logically equivalent.

Example 6.1. Consider again the two programs given earlier in this section. The answer theories for these two programs happen to be their respective Clark com-

[5] Notice that the set of stable models for the first program is $\{\{F, R\}, \{Q\}\}$, but for the second program it is $\{\{Q, R\}, \{F\}\}$. So these two programs are not equivalent in terms of their stable model semantics. Later we shall show that in the propositional case, our notion of equivalence coincides with that under the stable model semantics.

[6] $M, \sigma \models (\exists s)G[s]$ means that the formula $(\exists s)G[s]$ is true under the variable assignment σ in M.

[7] We remark here that equivalence between two programs under Wallace's semantics [31] can be similarly defined.

pletions. The first answer theory is

$$(\exists s)F(s) \equiv \neg(\exists s)Q(s) \wedge$$
$$(\exists s)R(s) \equiv (\exists s)F(s).$$

The second answer theory is

$$(\exists s)F(s) \equiv \neg(\exists s)Q(s) \wedge$$
$$(\exists s)R(s) \equiv (\exists s)Q(s).$$

These two theories are not equivalent.

According to Corollary 5.1 that relates our semantics to the stable model semantics, we see, by virtue of Proposition 6.1, that a stable model of a logic program P is a Herbrand model (in the sense of Corollary 5.1) of its answer theory. In particular, in the propositional case, the answer theory of a logic program is simply the disjunction of its stable models; and two programs P and P' are equivalent iff their stable models are the same:

Proposition 6.2. Let P and P' be two propositional logic programs.

1. *A theory T is the answer theory of P iff the set of models of T equals the set of stable models of P.*
2. *P and P' are equivalent iff the set of stable models of P equals the set of stable models of P'.*

The following proposition is straightforward:

Proposition 6.3. Let P and P' be two logic programs, and $\mathcal{D}$ and $\mathcal{D}'$ their respective action theories. The answer theory of P entails that of P' iff for any model M of $\mathcal{D}$, there is a model of M' of $\mathcal{D}'$ such that $M \sim M'$.

6.2. Equivalence of Definite Logic Programs

A logic program is *definite* if it does not mention any negative atoms in any of its clauses. The conditions for two definite programs to be equivalent are just as one would expect: P and P' are equivalent iff P entails each clause in P' and vice versa. In our language, we have:

Theorem 6.1. Let P and P' be two definite logic programs, and $\mathcal{D}$ and $\mathcal{D}'$ their action theories. P and P' are equivalent if and only if the following two conditions hold:

1. *For every clause $F(\vec{x})$:- G in P',*
$$\mathcal{D} \models (\forall \vec{x}).(\exists \vec{y}, s)G[s] \supset (\exists s)F(\vec{x}, s), \tag{6.1}$$
where $\vec{y}$ is the tuple of those variables in G but not in $\vec{x}$.
2. *Symmetrically, for every clause $F(\vec{x})$:- G in P,*
$$\mathcal{D}' \models (\forall \vec{x}).(\exists \vec{y}, s)G[s] \supset (\exists s)F(\vec{x}, s). \tag{6.2}$$

Proof. The "only if" follows directly from the definition. We show the "if" part.
Given any definite logic program P_1, and any model M of $\Sigma \cup \mathcal{D}_{una}$, there is a "unique" way of transforming M into a model of the action theory of P_1. Briefly,

this is done as follows. Let M' agree with M on everything except possibly on their interpretations of fluents. (So in particular, M and M' share the same domain for every sort.) Then M' is also a model of $\Sigma \cup \mathcal{D}_{una}$.

For any fluent $F(\vec{x}, s)$, let $M' \models (\forall x)\neg F(\vec{x}, S_0)$. Inductively, if the successor state axiom for F w.r.t. P_1 is:

$$(\forall \vec{x}, a, s) F(\vec{x}, do(a, s)) \equiv \Phi(a, \vec{x}, s),$$

then let

$$M' \models (\forall \vec{x}, a, s) F(\vec{x}, do(a, s)) \equiv \Phi(a, \vec{x}, s).$$

This is well-defined, because P_1 is a definite program, so $\Phi(a, \vec{x}, s)$ can be evaluated using the truth values of fluents in the situation s. In particular, it does not contain formulas like $(\forall s')F(s')$. Now it follows directly from the construction that M' is a model of the action theory for P_1.

Now we show the "if" part using Proposition 6.3. Suppose M is a model of $\mathcal{D}$. Let M' be the model of $\mathcal{D}'$ constructed from M as above. We show $M \sim M'$, i.e., by Proposition 6.1, for any fluent F and any variable assignment σ,

$$M, \sigma \models (\exists s)F(\vec{x}, s) \text{ iff } M', \sigma \models (\exists s)F(\vec{x}, s).$$

Suppose

$$M, \sigma \models F(\vec{x}, s).$$

We show by induction on s that $M', \sigma \models (\exists s)F(\vec{x}, s)$.

The case for $s = S_0$ is trivial. Inductively, suppose that for any fluent, this is true for s^*. We show that for any action α, this is true for $do(\alpha, s^*)$ as well. Since M is a model of $\mathcal{D}$, by the form of successor state axioms, there are two cases:

1. $M, \sigma \models F(\vec{x}, s^*)$.
2. α names one of the clauses in the definition of F in P, say

   $$F(\vec{x}) \ \texttt{:-}\ F_1(\vec{t_1}) \ \texttt{\&}\ F_2(\vec{t_2})$$

   and

   $$M, \sigma \models (\exists \vec{y}).F_1(\vec{t_1}, s^*) \wedge F_2(\vec{t_2}, s^*),$$

   where $\vec{y}$ is the tuple of variables mentioned in $\vec{t_1}$ or $\vec{t_2}$, but not in $\vec{x}$.

In the first case, the result follows from our inductive assumption. Suppose it is the latter case. By the inductive assumption, we have

$$M', \sigma \models (\exists \vec{y})[(\exists s')F_1(\vec{t_1}, s') \wedge (\exists s')F_2(\vec{t_2}, s')].$$

But M' is a model of $\mathcal{D}'$. So, by the assumption (6.2), instantiated to the above clause in P, we have:

$$M', \sigma \models (\exists \vec{y})[(\exists s')F_1(\vec{t_1}, s') \wedge (\exists s')F_2(\vec{t_2}, s')] \supset (\exists s')F(\vec{x}, s).$$

Therefore

$$M', \sigma \models (\exists s)F(\vec{x}, s).$$

This proves the inductive step. Therefore whenever $M, \sigma \models (\exists s)F(\vec{x}, s)$, we have $M', \sigma \models (\exists s)F(\vec{x}, s)$. The converse can be proved similarly. Thus $M \sim M'$.

Symmetrically, if $M \models \mathcal{D}'$, then there is a model M' of $\mathcal{D}$ such that $M \sim M'$.

This proves that P and P' are equivalent.

In principle, checking conditions (6.1) and (6.2) requires induction. However, there are some sufficient ways to do this that may be useful in practice. To illustrate the ideas, suppose the program P' contains the following clause:

$$F(x) \texttt{ :- } x = f(y) \texttt{ \& } F'(x, g(y))$$

and we want to verify (6.1) for it. Let a be a fresh constant symbol not already mentioned in P and P'. Then one way to do this is to first add $F'(f(a), g(a))$ to P, and then query the new program with $F(f(a))$. If the query succeeds, then the condition holds. However, if the query fails, this does not mean that the condition is false. For instance, this condition holds trivially when P is the empty program but G in (6.1) has at least one atom, because when P is empty, its action theory entails $\neg(\exists s)F(s)$ for any fluent F. This strategy was first used by Sagiv [23] for proving the equivalence of two deductive databases.

Yet another way to prove condition (6.1) say, is to query the program P with the goal G, and then query P again with the goal $F(\vec{x})$ for those bindings returned by the first query. If the goal $F(\vec{x})$ succeeds for all these bindings, then condition (6.1) holds. The problem with this strategy is that it may not work when there are infinitely many bindings for the first query.

Finally, we remark that if condition (6.1) holds, then $\mathcal{D}' \models (\exists s)F(\vec{t}, s)$ implies $\mathcal{D} \models (\exists s)F(\vec{t}, s)$, i.e. the set of ground atoms provable from P' is a subset of that from P. However, this does not mean that the answer theory of P will entail that of P'. For the latter to be true, the set of negative ground atoms provable from P' would have to be a subset of that from P as well. In fact, if the answer theory of a positive program entails that of another positive program, then these two positive programs must be equivalent. This is because for any positive program P, the models of the answer theory $\mathcal{D}$ of P must be unique in the sense that if two models of $\mathcal{D}$ share the same domains, then they must be the same. That in turn is because of the closed world assumption made in logic programs.

6.3. Normal Logic Programs

Unfortunately, Theorem 6.1 fails for logic programs with negation: Suppose the program P consists of the following single clause:

```
F:-
```

and the program P' consists of the following single clause:

```
F :- not F
```

Clearly, P and P' are not equivalent. But condition (6.1), which in this case is

$$\mathcal{D} \models \neg(\exists s)F(s) \supset (\exists s)F(s),$$

holds since $\mathcal{D} \models (\exists s)F(s)$, and condition (6.2) holds trivially because $\mathcal{D}'$ in this case is an inconsistent theory.

Intuitively, the reason Theorem 6.1 works for definite programs is that if $F(s)$ is provable from a definition program, then there must be a situation s' such that s' is *earlier* than s ($s' < s$), and a goal G such that $G[s']$ is provable and *does not* quantify over situations, i.e. $G[s']$ is a statement whose truth value can be determined by looking at the situation s' alone. This will ensure that there are no cycles, and thus induction on situation will work. This property is lost on normal logic programs because if G mentions negation, then $G[s]$ will quantify over situations, and its truth value will depend on the entire space of situations. To overcome this, the trick is then to introduce some new situation independent predicates, and replace formulas of the form $(\exists s')F(s')$ in $G[s]$ by atoms made of such new predicates. This will make the resulting formula "look" like a statement whose truth value depends only on the situation s. This is exactly the intuition behind the following definitions which will be used to formulate some sufficient conditions for two normal logic programs to be equivalent.

Suppose P is a logic program. For any predicate $F(\vec{x})$ in P, suppose $\widehat{F}(\vec{x})$ is a new predicate symbol with the same arity as F. We define the *loosened* action theory $\widehat{\mathcal{D}}$ of P to be that obtained from P's action theory $\mathcal{D}$ by replacing in $\mathcal{D}$'s successor state axioms every subformula of the form $(\exists s')F(\vec{t}, s')$ by $\widehat{F}(\vec{t})$, for every fluent F. For example, if the successor state axiom for F is

$$F(\vec{x}, do(a,s)) \equiv \\ \quad (\exists \vec{y})[a = A(\vec{x}) \wedge F_1(\vec{t_1}, s) \wedge \neg(\exists s')F_2(\vec{t_2}, s')] \vee F(\vec{x}, s),$$

then the new axiom, called the *loosened* successor state axiom of F, is

$$F(\vec{x}, do(a,s)) \equiv \\ \quad (\exists \vec{y})[a = A(\vec{x}) \wedge F_1(\vec{t_1}, s) \wedge \neg\widehat{F_2}(\vec{t_2})] \vee F(\vec{x}, s).$$

Loosened action theories are like action theories for definite logic programs. The propositions and theorems in Section 4 can be extended to them as well. For instance, corresponding to Theorem 4.1, we have: If $(\forall \vec{x}).(\exists s)F(\vec{x}, s) \equiv \Phi(\vec{x})$ is the Clark completion of the fluent F in the program P, then

$$\widehat{\mathcal{D}} \models (\forall \vec{x}).(\exists s)F(\vec{x}, s) \equiv \Phi'(\vec{x}), \tag{6.3}$$

where $\widehat{\mathcal{D}}$ is the loosened action theory of P, and Φ' is the result of replacing in Φ every subformula of the form $\neg(\exists s)F'(\vec{t}, s)$ by $\widehat{F'}(\vec{t})$, for every fluent F'.

In the following, we call a formula $\Phi(\vec{x}, s)$ a *simple state formula* if:

1. Its free variables are among $\vec{x}, s$.
2. It does not mention the initial state S_0.
3. It does not quantify over situations.
4. It does not mention *do*, *Poss*, or $<$.

Informally, $\Phi(\vec{x}, s)$ is a simple state formula when its truth value can be determined by the truth values of fluents in the situation s, and the truth values of state independent predicates like $\widehat{F}$ and equality.

In the following, we let $\mathcal{F}$ be:

$$\mathcal{F} = \{(\forall \vec{x})[(\exists s)F(\vec{x}, s) \equiv \widehat{F}(\vec{x})] \mid F \text{ is a fluent}\} \tag{6.4}$$

Theorem 6.2. Let P_1 and P_2 be two logic programs and let $\mathcal{D}_1$ and $\mathcal{D}_2$ be their respective action theories. In addition, let $\widehat{\mathcal{D}_2}$ be the loosened action theory of P_2. Finally, suppose

1. *For every clause $F(\vec{x})$ `:-` G in P_2,*

$$\mathcal{D}_1 \models (\forall \vec{x}).(\exists \vec{y}, s)G[s] \supset (\exists s)F(\vec{x}, s), \tag{6.5}$$

where $\vec{y}$ is the tuple of variables mentioned in G but not in $\vec{x}$.

2. *For every fluent $F(\vec{x}, s)$, there is a simple state formula $\Psi_F(\vec{x}, s)$ such that:*

(a) *In this formula, every fluent appears positively;*

(b) $\mathcal{D}_1 \cup \mathcal{F} \models (\forall \vec{x}).F(\vec{x}, s) \supset (\exists s')(s' < s \wedge \Psi_F(\vec{x}, s'))$; (6.6)

(c) $\widehat{\mathcal{D}_2} \models (\forall \vec{x}).\Psi'_F(\vec{x}) \supset (\exists s)F(\vec{x}, s)$, (6.7)

where $\Psi'_F(\vec{x})$ *is the result of replacing every atomic formula of the form* $F'(\vec{t}, s)$ *in* $\Psi_F(\vec{x}, s)$ *by* $(\exists s)F'(\vec{t}, s)$*, for every fluent* F'*.*

Then the answer theory of P_1 *entails that of* P_2*.*

Let us briefly comment on the theorem before proving it. Condition (6.6) means that if $F(s)$ holds, then there must be a situation s' *earlier* than s such that $\Psi_F(s')$ holds. Since Ψ_F is a simple state formula, its truth value depends only on s', so induction on situations will go through. Compared to Theorem 6.1, Ψ_F is like G. But since F :- G is a clause in P, condition (6.6) always holds for positive logic programs. Now condition (6.7) is like condition (6.2) in Theorem 6.1, with $\mathcal{D}'$ replaced by $\widehat{\mathcal{D}_2}$, and G by Ψ'_F.

PROOF. We need to show that for any model M_1 of $\mathcal{D}_1$, there is a model M_2 of $\mathcal{D}_2$ such that $M_1 \sim M_2$.

Suppose that M_1 is a model of $\mathcal{D}_1$. Since for any fluent F, $\widehat{F}$ does not appear in $\mathcal{D}_1$, we can assume that M_1 satisfies $\mathcal{F}$ as well.

Construct a first-order structure M_2 as follows:

1. It shares with M_1 the domain for every sort.
2. Except possibly on fluents, it shares with M_1 the interpretation of all function and predicate symbols. In particular, they agree on the new predicates.
3. For every fluent F, $M_2 \models (\forall \vec{x})\neg F(\vec{x}, S_0)$, and inductively, if

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

   is the loosened successor state axiom of F for P_2, then let

   $M_2 \models (\forall \vec{x}, a, s)[F(\vec{x}, do(a, s)) \equiv \Phi_F(\vec{x}, a, s)]$.

   Again (cf. the proof of Theorem 6.1), this is well defined because Φ_F is a simple state formula.

Clearly, M_2 is a model of $\widehat{\mathcal{D}_2}$, the loosened action theory of P_2.

We now show $M_1 \sim M_2$, i.e., for any fluent F, and any variable assignment σ,

$M_1, \sigma \models (\exists s)F(\vec{x}, s)$ iff $M_2, \sigma \models (\exists s)F(\vec{x}, s)$.

Suppose

$$M_2, \sigma \models F(\vec{x}, s).$$

We show by induction on s that $M_1, \sigma \models (\exists s)F(\vec{x}, s)$.

The case for $s = S_0$ is trivial. Inductively, suppose that for any fluent, this is true for s^*. We show that for any action α, this is true for $do(\alpha, s^*)$ as well. By the construction of M_2, and the form of successor state axioms, there are two cases:

1. $M_2, \sigma \models F(\vec{x}, s^*)$.
2. α names one of the clauses in the definition of F in P', say

   $$F(\vec{x}) \texttt{ :- } F_1(\vec{t_1}) \texttt{ \& not } F_2(\vec{t_2})$$

   and

   $M_2, \sigma \models (\exists \vec{y}).F_1(\vec{t_1}, s^*) \wedge \neg \widehat{F_2}(\vec{t_2})$, where $\vec{y}$ is the tuple of variables in $\vec{t_1}$ or $\vec{t_2}$, but not in $\vec{x}$.

In the first case, the result follows from our inductive assumption. So consider the latter case. By the inductive assumption, we have

$$M_1, \sigma \models (\exists \vec{y})[(\exists s')F_1(\vec{t_1}, s') \wedge \neg \widehat{F_2}(\vec{t_2})].$$

But M_1 is a model of the set $\mathcal{F}$ (6.4), thus

$$M_1, \sigma \models (\exists \vec{y})[(\exists s')F_1(\vec{t_1}, s') \wedge \neg(\exists s')F_2(\vec{t_2}, s')].$$

But the assumption (6.5), when instantiated to the above clause, yields:

$$M_1, \sigma \models (\exists \vec{y})[(\exists s')F_1(\vec{t_1}, s') \wedge \neg(\exists s')F_2(\vec{t_2}, s')] \supset (\exists s')F(\vec{x}, s').$$

So we have

$$M_1, \sigma \models (\exists s)F(\vec{x}, s).$$

This proves the inductive step. Therefore whenever $M_2, \sigma \models (\exists s)F(\vec{x}, s)$, we have $M_1, \sigma \models (\exists s)F(\vec{x}, s)$.

Now suppose $M_1, \sigma \models F(\vec{x}, s)$. We show that $M_2, \sigma \models (\exists s)F(\vec{x}, s)$. Again, we do induction on s. The case S_0 is vacuous because the assumption is false. Inductively, assume that the result holds for any fluent, for any state $s < s^*$. We show that it is true for s^* as well. Suppose $M_1, \sigma \models F(\vec{x}, s^*)$. Since M_1 is a model of $\mathcal{D}_1$ and the set (6.4), by the assumption (6.6), we have

$$M_1, \sigma \models (\exists s)(s < s^* \wedge \Psi_F(\vec{x}, s)).$$

Now by the inductive assumption, the fact that M_2 and M_1 agree on everything else except possibly on fluents, and the assumption that every fluent appears positively in Ψ_F, we have that

$$M_2, \sigma \models \Psi'_F(\vec{x})$$

where $\Psi'_F(\vec{x})$ is as in (6.7). Therefore, by the assumption (6.7), we have

$$M_2, \sigma \models (\exists s)F(\vec{x}, s).$$

This completes the inductive proof. So we have proved that $M_1 \sim M_2$. Using this, the fact that M_1 is a model of the set $\mathcal{F}$, and the fact that M_1 and M_2 agree on all new predicates $\widehat{F}$, we conclude that M_2 is a model of $\mathcal{F}$ (6.4) as well. Thus, by the definition of the loosened action theory, M_2 is a model of $\mathcal{D}_2$. This concludes the proof of the theorem.

To use Theorem 6.2, one needs to select the appropriate formula Ψ_F, for every fluent F. The following are some candidates.

Let

$$(\exists s)F(\vec{x}, s) \equiv \Phi(\vec{x})$$

be the Clark completion of F (see Theorem 4.1) in P_2.

- Let $\Psi_F(\vec{x}, s)$ be the result of replacing in Φ every positive subformula of the form $(\exists s)F'(\vec{t}, s)$ by $F'(\vec{t}, s)$, and every negative subformula of the form $(\exists s)F'(\vec{t}, s)$ by $\widehat{F'}(\vec{t})$. Then Ψ_F satisfies the required syntactic conditions: it is a simple state formula, and every fluent in it appears positively. By (6.3), it also satisfies the condition (6.7). So condition (6.6) is the only one left to be checked.

- If Φ has a positive occurrence of $(\exists s)F_1(\vec{t}, s)$, and

  $(\exists s)F_1(\vec{x}, s) \equiv \Phi_1(\vec{x})$

  is the Clark completion of F_1, then first let Φ' be the result of replacing in Φ this positive occurrence of $(\exists s)F_1(\vec{t}, s)$ by $\Phi_1(\vec{t})$, and let $\Psi_F(\vec{x})$ be obtained from Φ' the same way as it is obtained from Φ above. Then Ψ_F again is a simple state formula, every fluent in it appears positively, and satisfies condition (6.7).
- The above procedure of obtaining Ψ_F can be iterated. Notice that this procedure is closely related to unfolding (see below), and also *regression* [30, 18, 21] in planning.

For example, given the following Clark completion for F:

$$(\exists s)F(s) \equiv \{(\exists x, s)F_1(x, s) \vee [(\exists s)F_2(s) \wedge \neg(\exists s)F_3(s)]\}$$

we obtain the following possible Ψ_F:

$$(\exists x)F_1(x, s) \vee [F_2(s) \wedge \neg\widehat{F_3}]$$

Now suppose

$$(\exists s)F_1(x, s) \equiv (\exists s)F_4(x, s) \vee (\exists y)((\exists s)F_2(s) \wedge \neg(\exists s')F_5(x, y, s'))$$

is the Clark completion of F_1. First eliminate $(\exists s)F_1(x, s)$ in the Clark completion of F:

$$\begin{aligned}(\exists s)F(s) \equiv \{&(\exists x)[(\exists s)F_4(x, s) \vee (\exists y)((\exists s)F_2(s) \wedge \neg(\exists s')F_5(x, y, s'))] \vee \\ &(\exists s)F_2(s) \wedge \neg(\exists s)F_3(s)\}.\end{aligned}$$

¿From this, we get another possible Ψ_F:

$$(\exists x)[F_4(x, s) \vee (\exists y)(F_2(s) \wedge \neg\widehat{F_5}(x, y))] \vee [F_2(s) \wedge \neg\widehat{F_3}],$$

which is logically equivalent to

$$(\exists x)[F_4(x, s) \wedge F_2(s)] \vee (\exists x, y)[F_4(x, s) \wedge \neg\widehat{F_5}(x, y)] \vee [F_2(s) \wedge \neg\widehat{F_3}].$$

We further illustrate the use of the theorem by proving the following simple equivalence:

Proposition 6.4. Let P_1 be a program, F_1 an atom, and G a goal. Let P_2 be the union of P_1 with the clause:

$$F_1 \texttt{ :- } F_1 \texttt{ \& } G$$

Then P_1 and P_2 are equivalent.

PROOF. Suppose the Clark completion for F_1 in P_1 is

$$(\exists s)F_1(s) \equiv \Phi$$

and $\Psi_{F_1}(s)$ is a simple state formula obtained from Φ as outlined above. Then the Clark completion for F_1 in P_2 is of the form:

$$(\exists s)F_1(s) \equiv [(\exists s)F_1(s) \wedge \varphi] \vee \Phi \tag{6.8}$$

We show that the answer theory of P_2 entails that of P_1. The converse is easier, and can be similarly proved.

Condition (6.5). Trivial, since every clause in P_1 is also a clause in P_2.

Conditions (6.7) and (6.6). For each fluent F, we use the Clark completion of F in P_1 to generate the formula Ψ_F as outlined above. As we mentioned, in this case, only Condition (6.6) needs to be proved. There are two cases. If F is different from F_1, then the Clark completion of F in both P_1 and P_2 is the same. Suppose it is $(\exists s)F(s) \equiv \Phi_F$. By Corollary 4.2, we have

$$\mathcal{D}_2 \models (\forall s).F(s) \supset (\exists s')(s' < s \wedge \Phi'_F(s)),$$

where $\mathcal{D}_2$ is the action theory of P_2, and Φ'_F is the result of replacing in Φ_F every positive subformula of the form $(\exists s)F'(\vec{t}, s)$ by $F'(\vec{t}, s)$. Now by the construction of Ψ_F, we have

$$\mathcal{D}_2 \cup \mathcal{F} \models (\forall s).F(s) \supset (\exists s')(s' < s \wedge \Psi_F(s)),$$

which is the condition (6.6).

For F_1, we need to check:

$$\mathcal{D}_2 \cup \mathcal{F} \models (\forall s).F_1(s) \supset (\exists s')(s' < s \wedge \Psi_{F_1}(s)). \tag{6.9}$$

Using Corollary 4.2 for P_2, and the Clark completion (6.8) of F_1 in P_2, we have

$$\mathcal{D}_2 \cup \mathcal{F} \models (\forall s).F_1(s) \supset (\exists s')\{s' < s \wedge [(F_1(s') \wedge G[s']) \vee \Psi_{F_1}(s')]\}. \tag{6.10}$$

Now assume that $\mathcal{D}_2 \cup \mathcal{F}$, and $F_1(s)$. Since $\neg F_1(S_0)$, by our foundational axioms for the situation calculus, there is a situation of the form $do(a, s') \leq s$ such that

$$F_1(do(a, s')) \wedge (\forall s^*)(s^* \leq s' \supset \neg F_1(s^*)).$$

By (6.10), this means that $\Psi_{F_1}(s')$ must hold. So we have (6.9).

6.4. Unfold/Fold

Unfold/fold (Tamaki and Sato [27]) are among the best known program transformation operators. Seki [25] shows that they preserve the well-founded semantics of (Van Gelder and Ross and Schlipf [29]). Using Theorem 6.2, we can show, rather straightforwardly, that unfold/fold also preserve our situation calculus semantics. We illustrate using unfolding. For ease of presentation, we consider only the propositional case. The following definition is adapted from (Seki [25]).

Let P be a logic program, and C a clause in P of the form:

$$F_1 \;\texttt{:-}\; F_2 \;\texttt{\&}\; G$$

where F_1 and F_2 are distinct atoms. Suppose that

$$\begin{array}{c} F_2 \;\texttt{:-}\; G_1 \\ \vdots \\ F_2 \;\texttt{:-}\; G_k \end{array}$$

are all of the clauses in the definition of F_2 in P. Let C_i, $1 \leq i \leq k$, be the result of replacing F_2 in C by G_i. Then the program $P' = (P - \{C\}) \cup \{C_1, ..., C_k\}$ is called an *unfolding* of P. The clause C is called the *unfolding clause.*

Proposition 6.5. If P' is an unfolding of P, then P and P' are equivalent.

PROOF. Suppose that the Clark completion of F_1 in P is of the form

$$(\exists s)F_1(s) \equiv [((\exists s)F_2(s) \wedge \varphi]) \vee \Phi]. \tag{6.11}$$

Then the Clark completion of F_1 in P' is of the form

$$(\exists s)F_1(s) \equiv [(\varphi_1 \wedge \varphi) \vee \cdots \vee (\varphi_k \wedge \varphi) \vee \Phi]. \tag{6.12}$$

The Clark completion of F_2 in both programs is of the form

$$(\exists s)F_2(s) \equiv [\varphi_1 \vee \cdots \vee \varphi_k]. \tag{6.13}$$

Notice that (6.12) is a consequence of (6.11) and (6.13).

We show that the answer theory of P entails that of P'.

Condition (6.5). We only need to show this for the new clauses C_i, $1 \leq i \leq k$ in P':

$$\mathcal{D} \models (\exists s)(G_i[s] \wedge G[s]) \supset (\exists s)F_1(s).$$

But $(\exists s)(G_i[s] \wedge G[s])$ is equivalent to $\varphi_i \wedge \varphi$ (see Proposition 4.2 and Theorem 4.1). So this follows from (6.11) and (6.13).

Conditions (6.7) and (6.6). We only need to show these two conditions for F_1, since the Clark completion for other fluents are the same for both programs (see the above proof of Proposition 6.4). We take $\Psi_{F_1}(s)$ to be the formula obtained from the Clark completion (6.12) for F_1 in P' as outlined following Theorem 6.2. We only need to prove (6.6):

$$\mathcal{D} \cup \mathcal{F} \models (\forall s).F_1(s) \supset (\exists s')(s' < s \wedge \Psi_{F_1}(s')).$$

Assume that $\mathcal{D} \cup \mathcal{F}$ and $F_1(s)$. By Corollary 4.2, and the Clark completion (6.11) for F_1 in P, there is a $s' \leq s$ such that

$$(F_2(s') \wedge G[s']) \vee \Phi'(s'),$$

where $\Phi'(s)$ is the result of dropping $(\exists s)$ from all the positive occurrence of the subformula $(\exists s)F(s)$ in Φ. ¿From $F_2(s')$, by Corollary 4.2 for F_2, there is a $s^* < s'$ such that

$$\{(G_1[s^*] \vee \cdots \vee G_k[s^*]) \wedge G[s']\} \vee \Phi'(s').$$

Now by Corollary 4.1, we have

$$\{(G_1[s'] \vee \cdots \vee G_k[s']) \wedge G[s']\} \vee \Phi'(s').$$

By the definition of $\Psi_{F_1}(s)$, the above formula is equivalent to $\Psi_{F_1}(s')$ under the assumption that $\mathcal{F}$. This proves the condition (6.6), thus the answer theory of P entails that of P'.

The converse has a much easier proof, with the same formula $\Psi_{F_1}(s)$ for conditions (6.7) and (6.6). We do it for condition (6.6), because this is the only nontrivial one. We need to show

$$\mathcal{D}' \cup \mathcal{F} \models (\forall s).F_1(s) \supset (\exists s')(s' < s \wedge \Psi_{F_1}(s')).$$

Assume that $\mathcal{D}' \cup \mathcal{F}$ and $F_1(s)$. By Corollary 4.2 for F_1 in P', there is a $s' \leq s$ such that

$$(G_1[s'] \wedge G[s']) \vee \cdots \vee (G_k[s'] \wedge G[s']) \vee \Phi'(s').$$

Again by the definition of $\Psi_{F_1}(s)$, the above formula is equivalent to $\Psi_{F_1}(s')$ under the assumption that $\mathcal{F}$. This proves condition (6.6).

This proof generalizes to the first order case; we omit the details.

7. Other Applications

The framework of this paper is very general; it can be used to formalize many other aspects of logic programming languages.

Like most work on the formal semantics of logic programs, we have ignored many "dirty aspects" of the language, such as the cut operator. As mentioned earlier, one of the advantages of treating rules as actions is that we can reason about them as first-order objects within the logic. This is particularly useful in formalizing many search control operators in logic programming. As an example, we have formalized the cut (`!`) operator in Prolog using the basic framework proposed here ([10]).

Briefly, given a definite logic program P that contains cut, we proceed as follows to provide a semantics for P. First, we ignore cut, and delete all occurrences of `!` in P. This will give us a program that does not mention `!`, so the theory of this paper will be applicable, and an action theory $\mathcal{D}$ for it can be constructed. As we emphasized before, in $\mathcal{D}$, situations are derivation histories. However, due to the presence of `!`, some situations may not be reachable. A logical characterization of cut is then achieved by adding to $\mathcal{D}$ a situation calculus sentence that axiomatizes the set of reachable situations. We show that this semantics is well-behaved when the logic program is properly stratified. Furthermore, according to this semantics, the usual implementation of the negation-as-failure operator using cut is provably correct with respect to the stable model semantics. For details see [10].

We are also currently exploring the possibility of formalizing the dynamic "assert" and "retract" operators of Prolog within this framework. It is particularly interesting that once we allow "retract", the resulting theories of actions become much richer in that some actions will now have negative effects on fluents, and issues such as goal interactions in planning become relevant.

8. Concluding Remarks

By taking seriously the idea that rules are actions, we have formalized the declarative meaning of logic programs in the situation calculus. Like Clark's completion, our situation calculus semantics is formulated in classical logic. Unlike Clark's completion, our semantics is strong enough to handle recursion. Having a classical logical semantics has many advantages, one of which is the relative ease of proving properties of programs. To illustrate this, we have formulated conditions for two logic programs to be equivalent, and used them to prove the correctness of the unfolding transformation of (Tamaki and Sato [27]).

We have also used this framework to formalize various search control operators, and are working on extending it to the dynamic "assert" and "retract" operators of Prolog.

Acknowledgements

Our thanks to the other members of the University of Toronto Cognitive Robotics Group (Yves Lespérance, Hector Levesque, and Daniel Marcu) for their ideas, suggestions, and comments. Thanks also to G. Neelakantan Kartha, and Vladimir Lifschitz for helpful comments on an earlier draft of this paper. Our special thanks to Vladimir for bringing to our attention the work of Wallace [31]. This research was supported by grants from the National Science and Engineering Research Council of Canada, the Institute for Robotics and Intelligent Systems of Canada, and the Information Technology Research Center of Ontario.

REFERENCES

1. K. L. Clark. Negation as failure. In H. Gallaire and J. Minker, editors, *Logics and Databases*, pages 293–322. Plenum Press, New York, 1978.
2. M. Fitting. A Kripke-Kleene semantics for logic programs. *Journal of Logic Programming*, 2(4):295–312, 1985.
3. M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *Proc. Fifth International Conference and Symposium on Logic Programming*, pages 1070–1080, 1988.
4. 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.
5. 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, San Jose, CA, 1987. Morgan Kaufmann Publishers, Inc.
6. K. Kunen. Negation in logic programming. *Journal of Logic Programming*, 4(4):289–308, 1987.
7. H.J. Levesque, R. Reiter, Y. Lespérance, F. Lin and R. Scherl. GOLOG: A logic programming language for dynamic domains. *Journal of Logic Programming*, this volume.
8. V. Lifschitz. Pointwise circumscription. In *Proceedings of the Fifth National Conference on Artificial Intelligence (AAAI–86)*, pages 406–410, Philadelphia, PA, 1986.
9. V. Lifschitz. Formal theories of action. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence (IJCAI–87)*, pages 966–972, 1987.
10. F. Lin. A situation calculus semantics for the prolog cut operator. http://www.cs.toronto.edu/~cogrobo/, Draft. 1995.
11. F. Lin and R. Reiter. State constraints revisited. *Journal of Logic and Computation, Special Issue on Actions and Processes*, 4(5):655–678, 1994.
12. F. Lin and R. Reiter. Forget it! In R. Greiner and D. Subramanian, editors, *Working Notes of AAAI Fall Symposium on Relevance*, pages 154–159. The American Association for Artificial Intelligence, Menlo Park, CA, November 1994.
13. F. Lin and Y. Shoham. Provably correct theories of action. *Journal of the ACM*, 42(2):293–320, 1995.
14. J. W. Lloyd. *Foundations of Logic Programming*. Springer-Verlag, 2nd edition, 1987.
15. J. McCarthy. Situations, actions and causal laws. In M. Minsky, editor, *Semantic Information Processing*, pages 410–417. MIT Press, Cambridge, Mass., 1968.
16. J. McCarthy. Applications of circumscription to formalizing commonsense knowledge. *Artificial Intelligence*, 28:89–118, 1986.
17. 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, 1969.
18. E. P. Pednault. Synthesizing plans that contain actions with context-dependent effects. *Computational Intelligence*, 4:356–372, 1988.
19. E. P. Pednault. ADL: Exploring the middle ground between STRIPS and the situation calculus. In *Proceedings of the First International Conference on Princi-*

ples of Knowledge Representation and Reasoning (KR'89), pages 324–332. Morgan Kaufmann Publishers, Inc., 1989.
20. T. C. Przymusinski. On the declarative semantics of deductive databases and logic programs. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 193–216. Morgan Kaufmann, Los Altos, CA, 1988.
21. 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.
22. R. Reiter. Proving properties of states in the situation calculus. *Artificial Intelligence*, 64:337–351, 1993.
23. Y. Sagiv. Optimizing datalog programs. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 659–698. Morgan Kaufmann Publishers, San Mateo, CA., 1988.
24. 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.
25. H. Seki. Unfold/fold transformation of general logic programs for the well-founded semantics. *The Journal of Logic Programming*, 15:5–23, 1993.
26. Y. Shoham. Chronological ignorance: experiments in nonmonotonic temporal reasoning. *Artificial Intelligence*, 36:279–331, 1988.
27. H. Tamaki and T. Sato. Unfold/fold transformation of logic programs. In *Proceedings of the 2nd International Conference on Logic Programming*, pages 127–138, 1984.
28. A. Van Gelder. Negation as failure using tight derivations for general logic programs. *Journal of Logic Programming*, 6(2):109–133, 1989.
29. A. Van Gelder, K. A. Ross, and J. S. Schlipf. Unfounded sets and well-founded semantics for general logic programs. In *Proc. Seventh ACM Symposium on Principles of Database Systems*, pages 221–230, 1988.
30. R. Waldinger. Achieving several goals simultaneously. In E. Elcock and D. Michie, editors, *Machine Intelligence*, pages 94–136. Ellis Horwood, Edinburgh, Scotland, 1977.
31. M. G. Wallace. Tight, consistent, and computable completions for unrestricted logic programs. *Journal of Logic Programming*, 15:243–273, 1993.