Situation Calculus as Answer Set Programming

Joohyung Lee and **Ravi Palla**
School of Computing, Informatics and Decision Systems Engineering
Arizona State University
Tempe, AZ, 85287, USA
{joolee, Ravi.Palla}@asu.edu

Abstract

We show how the situation calculus can be reformulated in terms of the first-order stable model semantics. A further transformation into answer set programs allows us to use an answer set solver to perform propositional reasoning about the situation calculus. We also provide an answer set programming style encoding method for Reiter's basic action theories, which tells us how the solution to the frame problem in answer set programming is related to the solution in the situation calculus.

Introduction

The situation calculus (McCarthy & Hayes 1969; Reiter 2001) is one of the most well-known formalisms for reasoning about actions. Its language is first-order logic, sometimes enriched with second-order features. Prolog can be used to implement the situation calculus, based on the fact that Clark's completion semantics accounts for definitional axioms. This allows expressive first-order reasoning about actions.

On the other hand, there has been significant progress in efficient propositional reasoning thanks to the emergence of SAT solvers. SAT-based planning (Kautz & Selman 1992) was shown to be a competitive approach in planning. The idea was extended to action formalisms that are more expressive than STRIPS, such as nonmonotonic causal theories (Giunchiglia *et al.* 2004), which led to the implementation of the Causal Calculator [1] and the discrete event calculus (Mueller 2004), which led to the implementation of the DEC reasoner [2]. Answer set programming (ASP) is being widely applied, in part due to the availability of many efficient answer set solvers, which use SAT solvers or techniques adapted from SAT.

The input languages of such systems allow variables, the meaning of which is understood in terms of *grounding*—a process that replaces every variable with every variable-free (a.k.a. ground) term in the Herbrand universe. Under this approach, the domain to be modelled is assumed to be given and finite. Function symbols are usually disallowed as they introduce infinite ground terms (hence an infinite domain).

[1] http://www.cs.utexas.edu/users/tag/cc/ .

[2] http://decreasoner.sourceforge.net/ .

However, in the situation calculus, situations are treated as objects that can be quantified over, and function do takes actions and situations as arguments: the term $do(a_m, do(a_{m-1}, do(\ldots, do(a_1, S_0))))$ represents the situation that is obtained by applying actions $a_1, ..., a_m$ sequentially in the situation S_0. How do we build a system that can ground such theories?

In this paper, we apply propositional reasoning methods to the situation calculus by turning the situation calculus into answer set programming. More specifically, we reformulate Lin's causal action theories (Lin 1995) and Reiter's basic action theories (Reiter 2001) in terms of the first-order stable model semantics (Ferraris, Lee, & Lifschitz 2007; 2010). Under the assumption that the domain is given and finite, a further transformation is applied to turn them into answer set programs. We use the combination of systems F2LP [3] and DLV-COMPLEX [4] to do reasoning with a bounded number of situations. For basic action theories, we also provide an ASP encoding method that adopts the solution to the frame problem in ASP, which tells us how the solutions to the frame problem in the two formalisms are related to each other.

Preliminaries

Review of General Stable Model Semantics

We follow the definition of a stable model from (Ferraris, Lee, & Lifschitz 2010), a journal version of (Ferraris, Lee, & Lifschitz 2007).

Let $\mathbf{p}$ be a list of distinct predicate constants $p_1, \ldots, p_n$ other than equality. Given a first-order formula F, recall that CIRC$[F; \mathbf{p}]$ is defined as the second-order formula

$$F \wedge \neg\exists \mathbf{u}((\mathbf{u} < \mathbf{p}) \wedge F(\mathbf{u}))$$

where $\mathbf{u}$ is a list of distinct predicate variables of the same length as $\mathbf{p}$, expression $\mathbf{u} < \mathbf{p}$ stands for a formula expressing that $\mathbf{u}$ is "stronger than" $\mathbf{p}$, as defined in (Lifschitz 1994), and $F(\mathbf{u})$ is the formula obtained from F by substituting the variables $\mathbf{u}$ for the constants $\mathbf{p}$. Similarly, SM$[F; \mathbf{p}]$ is defined as the second-order sentence

$$F \wedge \neg\exists \mathbf{u}((\mathbf{u} < \mathbf{p}) \wedge F^*(\mathbf{u})),$$

where formula $F^*(\mathbf{u})$ is defined recursively as follows:

[3] http://reasoning.eas.asu.edu/f2lp .

[4] http://www.mat.unical.it/dlv-complex .

- $p_i(\mathbf{t})^* = u_i(\mathbf{t})$ for any tuple $\mathbf{t}$ of terms;
- $F^* = F$ for any atomic F that does not contain members of $\mathbf{p}$;
- $(F \odot G)^* = (F^* \odot G^*),\ \odot \in \{\wedge, \vee\}$;
- $(F \to G)^* = (F^* \to G^*) \wedge (F \to G)$;
- $(QxF)^* = QxF^*,\ Q \in \{\forall, \exists\}$.

A model of F (in the sense of first-order logic) is *stable* (relative to the list $\mathbf{p}$ of *intensional* predicates) if it satisfies SM$[F; \mathbf{p}]$. Let $\sigma(F)$ be the signature consisting of the object, function and predicate constants occurring in F. If F contains at least one object constant, an Herbrand interpretation of $\sigma(F)$ that satisfies SM$[F; \mathbf{p}]$ where $\mathbf{p}$ is the list of all predicate constants occurring in F, is called an *answer set* of F. The answer sets of a logic program Π are defined as the answer sets of the FOL-representation of Π (i.e., the conjunction of the universal closures of implications corresponding to the rules).

Proposition 2 of (Kim, Lee, & Palla 2009) tells that for the class of "canonical theories," the stable model semantics coincides with circumscription. We say that an occurrence of a predicate constant in a formula F is *strictly positive* if that occurrence is not in the antecedent of any implication. For any list $\mathbf{p}$ of predicate constants and any formulas G and H, an implication $G \to H$ is called *canonical (relative to* $\mathbf{p}$) if

- every occurrence of every predicate constant from $\mathbf{p}$ in G is strictly positive in G, and
- every occurrence of every predicate constant from $\mathbf{p}$ in H is strictly positive in H.

For instance, $p(x) \to q(x)$ is a canonical implication relative to $\{p, q\}$, while its contraposition $\neg q(x) \to \neg p(x)$ is not.

A *canonical theory (relative to* $\mathbf{p}$*)* is the conjunction of the universal closures of canonical implications.

Proposition 1 *(Kim, Lee, & Palla 2009, Proposition 2) For any canonical theory F relative to $\mathbf{p}$,*

$$\text{CIRC}[F; \mathbf{p}] \leftrightarrow \text{SM}[F; \mathbf{p}]$$

is logically valid.

The stable model semantics can be extended to allow strong negation (a.k.a. classical negation) (Ferraris, Lee, & Lifschitz 2010). We distinguish between intensional predicates of two kinds, *positive* and *negative*, and assume that each negative intensional predicate has the form $\sim p$, where p is a positive intensional predicate and '$\sim$' is a symbol for strong negation. An interpretation of the underlying signature is *coherent* if the extent of every negative predicate $\sim p$ in it is disjoint from the extent of the corresponding positive predicate p.

System F2LP

Building on the result of (Cabalar, Pearce, & Valverde 2005), Lee & Palla [2009] define a translation that turns an arbitrary first-order formula under the stable model semantics into a logic program consisting of a finite set of rules of the form

$$A_1\ ;\ldots;\ A_k \leftarrow\ A_{k+1},\ldots,A_m, \mathit{not}\ A_{m+1},\ldots,\mathit{not}\ A_n,\ \mathit{not\ not}\ A_{n+1},\ldots,\mathit{not\ not}\ A_p$$

($0 \le k \le m \le n \le p$), where each A_i is an atomic formula.

Definition 1 (Translation F2LP**)** *1. Given a formula F and a list of intensional predicates $\mathbf{p}$, eliminate the quantifiers, possibly introducing new predicates, by the procedure given in Definition 1 from (Lee & Palla 2009).*

2. Add choice formulas $(q(\mathbf{x}) \vee \neg q(\mathbf{x}))$ for all non-intensional predicates q.

3. Turn the resulting quantifier-free formula into a logic program by applying the transformation rules from (Cabalar, Pearce, & Valverde 2005).

(Lee & Palla 2009) shows the correctness of the translation under the assumption that every positive (negative, respectively) occurrence of a formula $\exists x G(x)$ ($\forall x G(x)$, respectively) in F belongs to a subformula H of F such that H contains no strictly positive occurrence of any intensional predicate.

System F2LP is an implementation of the translation above, so that existing answer set solvers can be used for computing the answer sets (i.e., Herbrand stable models) of a first-order theory.

Lin's Causal Action Theories in ASP

Here we consider how to turn Lin's causal action theories (Lin 1995) into answer set programs. The language of causal action theories is many-sorted first-order logic, whose sorts are *situation*, *action*, *fluent*, *truth value* and *object*. As in that paper we understand expression $P(\mathbf{x}, s)$ where P is a fluent name, as shorthand for $\mathit{Holds}(P(\mathbf{x}), s)$. We do not consider functional fluents in this paper.

According to Lin [1995], a formula $\phi(s)$ is called a *simple state formula about s* if $\phi(s)$ does not mention *Poss*, *Caused* or any situation term other than possibly the variable s.

We assume that a description $\mathcal{D}$ consists of a finite number of the following sets of axioms. We often identify $\mathcal{D}$ with the conjunction of the universal closures of all axioms in $\mathcal{D}$. In the following, F, F_i are fluent names, A is an action name, V, V_i are truth values, s, s' are situation variables, $\phi(s)$ is a simple state formula about s, symbols a, a' are action variables, f is a variable of sort fluent, v is a variable of sort truth value, and $\mathbf{x}$, $\mathbf{x}_i$, $\mathbf{y}$, $\mathbf{y}_i$ are lists of variables.

- $\mathcal{D}_{caused}$ is a set of axioms of the form

  $$\mathit{Poss}(A(\mathbf{x}), s) \to (\phi(s) \to \mathit{Caused}(F(\mathbf{y}), V, \mathit{do}(A(\mathbf{x}), s)),$$

  (direct effects) and

  $$\phi(s) \wedge \mathit{Caused}(F_1(\mathbf{x}_1), V_1, s) \wedge \ldots \wedge \mathit{Caused}(F_n(\mathbf{x}_n), V_n, s) \to \mathit{Caused}(F(\mathbf{x}), V, s)$$

  (indirect effects).
- $\mathcal{D}_{poss}$ is a set of axioms of the form

  $$\mathit{Poss}(A(\mathbf{x}), s) \leftrightarrow \phi(s). \tag{1}$$

- $\mathcal{D}_{rest}$ is a set of axioms of the following forms.
  - The basic axioms:

    $$\mathit{Caused}(f, \mathit{true}, s) \to \mathit{Holds}(f, s)$$
    $$\mathit{Caused}(f, \mathit{false}, s) \to \neg \mathit{Holds}(f, s)$$

    $$\mathit{true} \neq \mathit{false} \wedge \forall v (v = \mathit{true} \vee v = \mathit{false}). \tag{2}$$

– The unique name assumptions for fluent and action names:

$$F_i(\mathbf{x}) \neq F_j(\mathbf{y}), \quad (i \neq j)$$
$$F_i(\mathbf{x}) = F_i(\mathbf{y}) \rightarrow \mathbf{x} = \mathbf{y}. \tag{3}$$

Similarly for action names.

– The *foundational axioms* for the discrete situation calculus: [5]

$$s \neq do(a, s), \tag{4}$$
$$do(a, s) = do(a', s') \rightarrow (a = a' \wedge s = s'), \tag{5}$$
$$\forall p(p(S_0) \wedge \forall a, s(p(s) \rightarrow p(do(a, s))) \rightarrow \forall s\, p(s)). \tag{6}$$

– The frame axiom:

$$\mathit{Poss}(a, s) \rightarrow (\neg\exists v \mathit{Caused}(f, v, do(a, s)) \rightarrow (\mathit{Holds}(f, do(a, s)) \leftrightarrow \mathit{Holds}(f, s))).$$

– Axioms for other domain knowledge: $\phi(s)$.

Lin's causal action theory is defined as

$$\text{CIRC}[\mathcal{D}_{caused};\ \mathit{Caused}] \wedge \mathcal{D}_{poss} \wedge \mathcal{D}_{rest}. \tag{7}$$

Translation into ASP

We first reformulate Lin's causal action theories in terms of the first-order stable model semantics. It is easy to check that the theory (7) is canonical relative to *Caused*.

Let $\mathcal{D}_{poss\rightarrow}$ be the set of axioms $\phi(s) \rightarrow \mathit{Poss}(A(\mathbf{x}), s)$ for each axiom (1) in $\mathcal{D}_{poss}$. Instead of the second-order axiom (6) we consider the following first-order formula, which introduces new intensional predicate *Sit*.[6]

$$\mathit{Sit}(S_0) \wedge \forall a, s(\mathit{Sit}(s) \rightarrow \mathit{Sit}(do(a, s))) \wedge \neg\exists s \neg \mathit{Sit}(s). \tag{8}$$

In the following $\mathcal{D}^{-}_{rest}$ is the theory obtained from $\mathcal{D}_{rest}$ by dropping (6).

Theorem 1 *The situation calculus domain description (7) is equivalent to each of the following when we disregard the auxiliary predicate Sit:*

(a) $\text{SM}[\mathcal{D}_{caused}; \mathit{Caused}] \wedge \mathcal{D}_{poss} \wedge \mathcal{D}^{-}_{rest} \wedge \text{SM}[(8); \mathit{Sit}]$*;*

(b) $\text{SM}[\mathcal{D}_{caused};\ \mathit{Caused}] \wedge \text{SM}[\mathcal{D}_{poss\rightarrow};\ \mathit{Poss}] \wedge \mathcal{D}^{-}_{rest} \wedge \text{SM}[(8); \mathit{Sit}]$*;*

(c) $\text{SM}[\mathcal{D}_{caused} \wedge \mathcal{D}_{poss\rightarrow} \wedge \mathcal{D}^{-}_{rest} \wedge (8);\ \mathit{Caused}, \mathit{Poss}, \mathit{Sit}]$.

The translation F2LP can be used to turn (c) into a logic program syntax.

Theorem 2 *Let $\mathcal{D}$ be a finite causal action theory (7) whose signature contains finitely many predicate constants* $\mathbf{p}$*, let F be $\mathcal{D}_{caused} \wedge \mathcal{D}_{poss\rightarrow} \wedge \mathcal{D}^{-}_{rest} \wedge (8)$ and let Π^{FOL} be the FOL representation of the program obtained by applying the translation* F2LP *on F in which only predicates Caused, Poss, Sit are assumed to be intensional. Then the models of* $\text{SM}[\Pi^{FOL}; \mathbf{p}, \mathit{Sit}, \mathbf{q}]$ *restricted to the signature of $\mathcal{D}$ are precisely the models of $\mathcal{D}$, where* $\mathbf{q}$ *is the list of all new predicates introduced by the translation.*

[5] For simplicity we skip two other axioms regarding the partial-order among situations.

[6] Suggested by Vladimir Lifschitz (personal communication).

In the context of answer set programming, since answer sets are Herbrand interpretations, we can drop axioms (2)–(5). Also we do not include (8) in the input to F2LP. Instead, we include the following set of rules $\Pi_{situation}$ (in the language of DLV-COMPLEX) to the program $\Pi_{description}$ that is obtained by running F2LP on the rest of the axioms.

```
nesting(0,s0).
nesting(L1,do(A,S)) :- nesting(L,S), #succ(L,L1),
                       action(A).
situation(S) :- nesting(L,S).
```

$\Pi_{situation}$ is used to generate finitely many situation terms whose height is up to `#maxint`, the value that can be given as an option in invoking DLV-COMPLEX. `#succ` is an arithmetic predicate in the language of DLV-COMPLEX defined over integers: `#succ(L,L1)` is true iff $L1 = L + 1$. It is not difficult to check that if a program Π containing these rules has no occurrence of predicate `nesting` in the other rules and no occurrence of predicate `situation` in the head of any other rule in Π, then every answer set of Π contains atoms $\texttt{situation}(\texttt{do}(\texttt{a}_\texttt{m}, \texttt{do}(\texttt{a}_{\texttt{m}-1}, \texttt{do}(\ldots, \texttt{do}(\texttt{a}_1, \texttt{s0})))))$ for all possible sequences of actions $\texttt{a}_1, \ldots, \texttt{a}_\texttt{m}$ for $\texttt{m} = 0, \ldots, \#\texttt{maxint}$. The use of predefined symbols like `#succ` and `#maxint` is not essential. $\Pi_{situation}$ can be written in the language of any other answer set solver by explicitly providing the integer sort and defining the successor relation. However, no other answer set solvers, to the best of our knowledge, can ground $\Pi_{situation}$. This is because the recursion in the second argument of `nesting` in the second rule leads to violation of the syntactic conditions, such as λ-restricted, ω-restricted, or finite domain programs, that answer set solvers impose on the input languages in order to ensure finite grounding in the presence of function symbols. Nonetheless, the rules can be finitely grounded by DLV-COMPLEX since it allows us to turn off the finite domain checking (option `-nofdcheck`). It is not difficult to see why the program leads to finite grounding since we provide an explicit upper limit for the function nesting.

Figure 1 shows the encoding of Lin's suitcase example (Lin 1995) in the language of F2LP. In order to turn the many sorted signature into the non-sorted one, we add "domain predicates" in the antecedents of axioms. Since we fix the domain of situations to be finite, we require that actions not be allowed to be effective in the final situations. This is done by introducing predicate `final(S)`.

Consider the simple temporal projection problem from (Lin 1995): Initially the first lock is down and the second lock is up. What will happen if the action flip turns the first lock up? We can check the answer using F2LP and DLV-COMPLEX: first, we add the following rules to the theory in Figure 1. The last rule is the negation of the conclusion we want to check.

```
% initial situation
:- h(up(l1),s0).
h(up(l2),s0).

% query
:- h(open,do(flip(l1),s0)).
```

```
value(t).   value(f).   lock(l1).   lock(l2).

fluent(up(X)) :- lock(X).
fluent(open).

action(flip(X)) :- lock(X).

% defining the situation domain
nesting(0,s0).
nesting(L1,do(A,S)) :- nesting(L,S), #succ(L,L1),
                       action(A).
situation(S) :- nesting(L,S).
final(S) :- nesting(L,S), L = #maxint.

% basic axioms
fluent(F) & situation(S) & caused(F,t,S) -> h(F,S).
fluent(F) & situation(S) & caused(F,f,S) -> -h(F,S).

% D_caused
lock(X) & situation(S) & -final(S)
  & poss(flip(X),S) ->
    (h(up(X),S) -> caused(up(X),f,do(flip(X),S))).
lock(X) & situation(S) & -final(S)
  & poss(flip(X),S) ->
    (-h(up(X),S) -> caused(up(X),t,do(flip(X),S))).

situation(S) & h(up(l1),S) & h(up(l2),S)
                              -> caused(open,t,S).

% D_poss
lock(X) & situation(S) -> poss(flip(X),S).

% frame axioms
action(A) & situation(S) & -final(S) & fluent(F)
  & poss(A,S) -> (-?[V]:caused(F,V,do(A,S)) ->
  ((h(F,do(A,S))->h(F,S)) & (h(F,S)->h(F,do(A,S))))).

% Holds is non-intensional
fluent(F) & situation(S) -> h(F,S) | -h(F,S).
```

Figure 1: Lin's Suitcase in the language of F2LP

We run F2LP on this theory to generate logic program `suitcase.lp` and then call DLV-COMPLEX as follows:

```
% dlv-complex -nofdcheck -N=1 suitcase.lp
```

"`-N=1`" sets the `#maxint` to be 1. DLV-COMPLEX returns no answer set as expected.

Lin [1995] provides a computing method for his action theories using completion. Since completion is weak to handle cycle and recursion, the limitation is noted in that paper. This is not the case with our approach, which relies on the stable model semantics. For instance, consider the Robby's apartment example from (Doğandağ, Ferraris, & Lifschitz 2004), in which Robby the robot has to find a plan to unlock the doors so that any room is accessible from any other. A part of the description shown below cannot be handled by completion, but can be handled in our approach.

$$\begin{aligned} &\mathit{Holds}(\mathit{Unlocked}(x,y),s) \rightarrow \mathit{Caused}(\mathit{Accessible}(x,y),true,s),\\ &\mathit{Holds}(\mathit{Unlocked}(x,y),s) \wedge \mathit{Caused}(\mathit{Accessible}(y,z),true,s)\\ &\qquad\qquad \rightarrow \mathit{Caused}(\mathit{Accessible}(x,z),true,s). \end{aligned} \tag{9}$$

Reiter's Basic Action Theories in ASP

As before, we understand $P(\mathbf{x},s)$ where P is a fluent name, as shorthand for $\mathit{Holds}(P(\mathbf{x}),s)$, and we do not consider functional fluents.

A *basic action theory (BAT)* is of the form

$$\Sigma \cup \mathcal{D}_{ss} \cup \mathcal{D}_{ap} \cup \mathcal{D}_{una} \cup \mathcal{D}_{S_0}, \tag{10}$$

where

- Σ is the set of the foundational axioms;
- $\mathcal{D}_{ss}$ is a set of successor state axioms of the form
$$F(\mathbf{x}, do(a,s)) \leftrightarrow \Phi_F(\mathbf{x},a,s),$$
where $\Phi_F(\mathbf{x},a,s)$ is a formula that is *uniform* in s (Reiter 2001) and whose free variables are among $\mathbf{x}, a, s$;
- $\mathcal{D}_{ap}$ is a set of action precondition axioms of the form
$$\mathit{Poss}(A(\mathbf{x}),s) \leftrightarrow \Pi_A(\mathbf{x},s),$$
where $\Pi_A(\mathbf{x},s)$ is a formula that is uniform in s and whose free variables are among $\mathbf{x}, s$;
- $\mathcal{D}_{una}$ is the set of unique name axioms for fluents and actions;
- $\mathcal{D}_{S_0}$ is a set of first-order sentences that are uniform in S_0.

Translation into ASP

In the following, $\mathcal{D}$ is a finite BAT (10) whose signature contains finitely many predicate constants $\mathbf{p}$, and $\mathcal{D}^-$ is a theory obtained from $\mathcal{D}$ by dropping (6).

Theorem 3 *Let Π^{FOL} be the FOL-representation of the program obtained by applying translation* F2LP *on $\mathcal{D}^- \wedge (8)$ in which only predicate Sit is assumed to be intensional. Then the models of* SM$[\Pi^{FOL}; \mathbf{p}, Sit, \mathbf{q}]$ *restricted to the signature of $\mathcal{D}$ are precisely the models of $\mathcal{D}$, where $\mathbf{q}$ is the list of all new predicates introduced by the translation.*

As with the case of Lin's causal action theories, in order to run Π^{FOL} using DLV-COMPLEX, we need to restrict the domain of situations to be finite. Also axioms (3)–(5) can be dropped since answer sets are Herbrand interpretations.

Alternative Encoding in ASP

In this section we consider an alternative encoding of BAT in ASP, in which we do not need to provide explicit successor state axioms $\mathcal{D}_{ss}$.

ASP-style BAT is of the form

$$\Sigma \cup \mathcal{D}_{effect} \cup \mathcal{D}_{precond} \cup \mathcal{D}_{S_0} \cup \mathcal{D}_{una} \cup \mathcal{D}_{inertia} \cup \mathcal{D}_{exogenous_0} \tag{11}$$

where

- Σ, $\mathcal{D}_{S_0}$, $\mathcal{D}_{una}$ are defined as before;
- $\mathcal{D}_{effect}$ is a finite set of axioms of the form
$$\gamma_R^+(\mathbf{x},a,s) \rightarrow \mathit{Holds}(R(\mathbf{x}), do(a,s)) \tag{12}$$
or
$$\gamma_R^-(\mathbf{x},a,s) \rightarrow \sim \mathit{Holds}(R(\mathbf{x}), do(a,s)), \tag{13}$$
where $\gamma_R^+(\mathbf{x},a,s)$ and $\gamma_R^-(\mathbf{x},a,s)$ are formulas that are uniform in s and whose free variables are among $\mathbf{x}, a$ and s;

- $\mathcal{D}_{precond}$ is a finite set of axioms of the form
$$\pi_A(\mathbf{x}, s) \rightarrow \mathit{Poss}(A(\mathbf{x}), s) \quad (14)$$
where $\pi_A(\mathbf{x}, s)$ is a formula that is uniform in s and whose free variables are among $\mathbf{x}, s$;
- $\mathcal{D}_{inertia}$ is the set of the axioms
$$\mathit{Holds}(R(\mathbf{x}), s) \wedge \neg \sim \mathit{Holds}(R(\mathbf{x}), do(a, s)) \rightarrow \mathit{Holds}(R(\mathbf{x}), do(a, s)),$$
$$\sim \mathit{Holds}(R(\mathbf{x}), s) \wedge \neg \mathit{Holds}(R(\mathbf{x}), do(a, s)) \rightarrow \sim \mathit{Holds}(R(\mathbf{x}), do(a, s))$$
for all fluent names R;
- $\mathcal{D}_{exogenous_0}$ is
$$\mathit{Holds}(R(\mathbf{x}), S_0) \vee \sim \mathit{Holds}(R(\mathbf{x}), S_0)$$
for all fluent names R.

Note that $\mathcal{D}_{inertia}$ are axioms used in answer set planning to handle the frame problem (Lifschitz & Turner 1999). Similarly, $\mathcal{D}_{exogenous_0}$ is used in ASP with strong negation to represent that the initial value of a fluent is arbitrary.

We will show how this is related to Reiter's BAT. Since we are going to use strong negation, it is convenient to define the following notion.

Let σ be a signature, let $\mathbf{p}$ be a set of predicate constants in σ, and let $\sigma^{\mathbf{p}}$ be a signature obtained from σ by adding $\sim p$ for all predicate constants p in $\mathbf{p}$. We say that an interpretation I of $\sigma^{\mathbf{p}}$ is *complete* on $\mathbf{p}$ if it satisfies
$$\forall \mathbf{x}(p(\mathbf{x}) \vee \sim p(\mathbf{x}))$$
for every p in $\mathbf{p}$.

Let I be a coherent interpretation of $\sigma^{\mathbf{p}}$ that is complete on $\mathbf{p}$. By $\widetilde{I}$ we denote the interpretation of σ that agrees with $\sigma^{\mathbf{p}}$ on all constants not in $\mathbf{p}$ and for each p in $\mathbf{p}$,

- $\mathbf{t}^{\widetilde{I}} \in p^{\widetilde{I}}$ if $\mathbf{t}^I \in p^I$
- $\mathbf{t}^{\widetilde{I}} \notin p^{\widetilde{I}}$ if $\mathbf{t}^I \in (\sim p)^I$.

Let $\mathcal{D}'_{ss}$ be the set of successor state axioms
$$Holds(R(\mathbf{x}), do(a, s)) \leftrightarrow \Gamma^+_R(\mathbf{x}, a, s) \vee (Holds(R(\mathbf{x}), s) \wedge \neg \Gamma^-_R(\mathbf{x}, a, s))$$
where $\Gamma^+_R(\mathbf{x}, a, s)$ is the disjunction of $\gamma^+_R(\mathbf{x}, a, s)$ for all axioms (12) in $\mathcal{D}_{effect}$, and $\Gamma^-_R(\mathbf{x}, a, s)$ is the disjunction of $\gamma^-_R(\mathbf{x}, a, s)$ for all axioms (13) in $\mathcal{D}_{effect}$.

By $\mathcal{D}'_{ap}$ we denote the set of axioms $\mathit{Poss}(A(\mathbf{x}), s) \leftrightarrow \Pi_A(\mathbf{x}, s)$ where $\Pi_A(\mathbf{x}, s)$ is the disjunction of $\pi_A(\mathbf{x}, s)$ for all axioms (14) in $\mathcal{D}_{precond}$.

In the following statement, for simplicity, we assume that Σ does not contain the second-order axiom (6).[7] Instead we consider only the models I such that the domain of situation is the smallest set $\mathcal{S}$ satisfying

- σ_0 is in $\mathcal{S}$, where $\sigma_0 = S_0^I$;
- if $\sigma \in \mathcal{S}$ and a is an element in the action domain, $do^I(a, \sigma)$ is in $\mathcal{S}$.

Theorem 4 *Let T be a theory (11) of signature σ^{Holds}. If I satisfies $\neg \exists \mathbf{x} as(\Gamma^+_R(\mathbf{x}, a, s) \wedge \Gamma^-_R(\mathbf{x}, a, s))$ for every fluent name R, then I satisfies* SM$[T; \mathit{Poss}, \mathit{Holds}, \sim \mathit{Holds}]$ *iff $\widetilde{I}$ satisfies the BAT $\Sigma \wedge \mathcal{D}'_{ss} \wedge \mathcal{D}'_{ap} \wedge \mathcal{D}_{una} \wedge \mathcal{D}_{S_0}$.*

[7] Alternatively, we can consider extending the notion SM$[F]$ to second-order formula F, or as before consider (8) instead.

Situation Calculus Planning in ASP

Not every situation is executable, but a plan is about executable situation. In (Reiter 2001), planning in situation calculus is described as follows: given a domain description $\mathcal{D}$ and a situation calculus formula $G(s)$ whose only free variable is s, a variable free situation term σ is a plan for G iff
$$\mathcal{D} \models \mathit{Executable}(\sigma) \wedge G(\sigma)$$
where $\mathit{Executable}(s)$ is an abbreviation of the formula
$$\forall a, s^*(do(a, s^*) \sqsubseteq s \rightarrow \mathit{Poss}(a, s^*)).$$
Such σ can be obtained as a side-effect of proving the sentence $\exists s(\mathit{Executable}(s) \wedge G(s))$.

We provide an alternative method tailored to answer set programming. We add to the domain description $\mathcal{D}$ the following axioms $\mathcal{D}_{executable}$,

$\mathit{Executable}(S_0)$,
$\mathit{Executable}(s) \wedge \mathit{Poss}(a, s) \wedge \neg \mathit{Final}(s) \rightarrow \mathit{Executable}(do(a, s))$.

and a goal $\mathcal{D}_{goal}$,
$$\neg\neg \exists s(\mathit{Executable}(s) \wedge G(s)).$$

Given the program $\Pi_{description}$ obtained by applying F2LP on the situation calculus description as explained earlier, the answer sets of $\Pi_{description} \cup \Pi_{situation} \cup \mathcal{D}_{executable} \cup \mathcal{D}_{goal}$ contain all plans of length up to `#maxint`. For instance, consider a simple planning problem to open the suitcase when the locks are initially down. We add the following axioms to Figure 1.

```
executable(s0).
executable(S) & poss(A,S) & -final(S)
  & situation(S) & action(A) -> executable(do(A,S)).

:- h(up(l1),s0).
:- h(up(l2),s0).
:- h(open,s0).

--?[S]: (executable(S) & h(open,S)).
```

When `#maxint` is 1, F2LP and DLV-COMPLEX find no answer set and when `#maxint` is 2, they find the unique answer set that contains both `h(open,do(flip(l2),do(flip(l1),s0)))` and `h(open,do(flip(l1),do(flip(l2),s0)))`, each of which encodes a plan. In other words, the single answer set contains multiple plans in different branches of the situation tree.

Related Work

Prolog provides a natural implementation for basic action theories since definitional axioms can be represented by Prolog rules according to the Clark's theorem (Reiter 2001, Chapter 5). The Lloyd-Topor transformation that is used to turn formulas into Prolog rules is similar to the translation F2LP, but they are different in that the former preserves the completion semantics and the latter the stable model semantics.

Gelfond & Lifschitz [1993] introduce action language $\mathcal{A}$ and provide a translation from $\mathcal{A}$ to logic programs based

on the situation calculus approach. Their work was followed by several others. Lin & Wang [1999] provide an approach to generate successor state axioms from a restricted set of "clausal" causal theories using answer set solvers. Our translation scheme is close to (Kim, Lee, & Palla 2009), which shows how to turn the event calculus into answer set programming. There the ASP-based event calculus reasoner was shown to be faster than the SAT-based DEC reasoner, thanks to efficient grounding methods implemented in ASP solvers. We expect that our ASP-based situation calculus reasoning is efficient as well. However, to the best of our knowledge, there is no other implementation that we can directly compare with in the propositional case.

Kautz & Selman [1992] introduce linear encodings that are similar to a propositionalized version of situation calculus (McCarthy & Hayes 1969). Lin [2003] introduces an action description language and describes a procedure that generates successor state axioms. The soundness of the procedure is shown with respect to a translation from action domain descriptions into his causal theories. (Claßen *et al.* 2007) proposes an integration of Golog with the state-of-the-art PDDL planner FF, and shows the performance improvements over the original Golog. Alternatively an ASP based situation calculus reasoner may substitute for FF, which would enable us to perform more expressive reasoning tasks than what Golog+FF can handle. It will be also interesting to compare the performance of Golog+ASP with Golog+FF.

Conclusion

In this paper, we show how Lin's causal theories and Reiter's basic action theories can be reformulated in terms of the first-order stable model semantics. Under the assumption that the domain is given and finite, system F2LP can be used to turn the resulting formulas under the stable model semantics into answer set programs so that DLV-COMPLEX can be used for computing the propositional models of the situation calculus description. In comparison with the computing method based on completion given in (Lin 1995), our ASP-based method can handle recursive axioms that cannot be handled by completion. Also we show an ASP-style encoding for Reiter's BAT, which tells us how the solution to the frame problem in ASP is related to the solution employed in BAT.

Our work shows that the first-order stable model semantics is useful in relating classical logic based action formalisms to answer set programming. The translations introduced in this paper would have been more complicated if we did not involve the first-order stable model semantics.

Acknowledgements: We are grateful to Vladimir Lifschitz for useful discussions related to this paper. This research was partially supported by the National Science Foundation under grants IIS-0916116 and by the Office of the Director of National Intelligence (ODNI), Intelligence Advanced Research Projects Activity (IARPA), through US army. All statements of fact, opinion or conclusions contained herein are those of the authors and should not be construed as representing the official views or policies of IARPA, the ODNI or the U.S. Government.

References

Cabalar, P.; Pearce, D.; and Valverde, A. 2005. Reducing propositional theories in equilibrium logic to logic programs. In *Proceedings of Portuguese Conference on Artificial Intelligence (EPIA)*, 4–17.

Claßen, J.; Eyerich, P.; Lakemeyer, G.; and Nebel, B. 2007. Towards an integration of golog and planning. In *Proc. IJCAI*, 1846–1851.

Doğandağ, S.; Ferraris, P.; and Lifschitz, V. 2004. Almost definite causal theories. In *Proc. LPNMR*, 74–86.

Ferraris, P.; Lee, J.; and Lifschitz, V. 2007. A new perspective on stable models. In *Proc. IJCAI*, 372–379.

Ferraris, P.; Lee, J.; and Lifschitz, V. 2010. Stable models and circumscription. [8] *Artificial Intelligence*. To appear.

Gelfond, M., and Lifschitz, V. 1993. Representing action and change by logic programs. *Journal of Logic Programming* 17:301–322.

Giunchiglia, E.; Lee, J.; Lifschitz, V.; McCain, N.; and Turner, H. 2004. Nonmonotonic causal theories. *Artificial Intelligence* 153(1–2):49–104.

Kautz, H., and Selman, B. 1992. Planning as satisfiability. In *Proceedings of European Conference on Artificial Intelligence (ECAI)*, 359–363.

Kim, T.-W.; Lee, J.; and Palla, R. 2009. Circumscriptive event calculus as answer set programming. In *Proc. IJCAI*, 823–829.

Lee, J., and Palla, R. 2009. System F2LP – computing answer sets of first-order formulas. In *Proc. LPNMR*, 515–521.

Lifschitz, V., and Turner, H. 1999. Representing transition systems by logic programs. In *Proc. LPNMR*, 92–106.

Lifschitz, V. 1994. Circumscription. *Handbook of Logic in AI and Logic Programming*, volume 3. Oxford University Press. 298–352.

Lin, F., and Wang, K. 1999. From causal theories to logic programs (sometimes). In *Proc. LPNMR*, 117–131.

Lin, F. 1995. Embracing causality in specifying the indirect effects of actions. In *Proc. IJCAI*, 1985–1991.

Lin, F. 2003. Compiling causal theories to successor state axioms and STRIPS-like systems. *Journal of Artificial Intelligence Research* 19:279–314.

McCarthy, J., and Hayes, P. 1969. Some philosophical problems from the standpoint of artificial intelligence. *Machine Intelligence*, volume 4. Edinburgh: Edinburgh University Press. 463–502.

Mueller, E. T. 2004. Event calculus reasoning through satisfiability. *Journal of Logic and Computation* 14(5):703–730.

Reiter, R. 2001. *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*. MIT Press.

[8] `http://peace.eas.asu.edu/joolee/papers/smcirc.pdf`.