Refining Action Theories through Abductive Logic Programming

Renwei Li[1*] Luis Moniz Pereira[1] Veronica Dahl[2]

[1] Department of Computer Science
Universidade Nova de Lisboa
2825 Monte de Caparica
Portugal
{renwei,lmp}@di.fct.unl.pt

[2] School of Computing Science
Simon Fraser University
Burnaby, B.C. V5A 1S6
Canada
veronica@cs.sfu.ca

Abstract

Reasoning about actions and changes often starts with an action theory which is then used for planning, prediction or explanation. In practice it is sometimes not simple to give an immediately available action theory. In this paper we will present an abductive methodology for describing action domains. We start with an action theory which is not complete, i.e., has more than one model. Then, after some tests are done, we can abduce a complete action theory. Technically, we use a high level action language to describe incomplete domains and tests. Then, we present a translation from domain descriptions to abductive logic programs. Using tests, we then abductively refine an original domain description to a new one which is closer to the domain in reality. The translation has been shown to be both sound and complete. The result of this paper can be used not only for refinement of domain descriptions but also for abductive planning, prediction and explanation. The methodology presented in this paper has been implemented by an abductive logic programming system.

1 Introduction

When reasoning about actions and changes, we often assume that an action theory has been given and described in a formal language or in a framework, e.g. situation calculus [15], event calculus [9], action description languages $\mathcal{A}$

*Renwei Li is currently visiting School of Computing Science, Simon Fraser University, Burnaby, B.C. V5A 1S6, Canada, renwei@cs.sfu.ca.

[6] and ADL [16], the fluent-features framework (FFF) [19], and their variants or extensions. But little work has been reported on how to obtain an action theory. Assume that we want to generate a plan to make the world in a definite state (goal), but we are not certain about the initial state and the effects of available actions. For example, let's consider Vladimir Lifschitz' challenge problem[1]:

> The room has two lamps, say Big and Small, and two light switches, say Left and Right. A switch controls one and only one light. Both lights are off. Initially we don't know whether the wiring is this way or the other way around, but we can find out by toggling a switch.

In this example, we have two actions: to toggle the left switch and to toggle the right switch, denoted by $toggle(left)$ and $toggle(right)$, and we have two fluents: the big light is on and the small light is on, denoted by $on(big)$ and $on(small)$. If we knew the way in which the circuit is connected, then we could generate plans, predict the future, or explain the past. The problem is that no such an immediately available theory is available. An intelligent agent should be able to perform some tests and then obtain a complete action theory. In this paper we will present an abductive methodology for reasoning about actions and changes starting from an incomplete action theory, i.e., an action theory with more than one model, then refining it by testing and abductive reasoning so as to have a complete action theory, which can then be used for planning, predicting and explaining depending on specific needs. Our methodology consists of a high-level action description language $\mathcal{A}^+$, a translation from $\mathcal{A}^+$ to abductive logic programs, and an abductive logic programming system used as the underlying inference engine for refinement.

Now suppose that we have an action description language obtained by extending $\mathcal{A}$ [6] with propositional conjunctions and disjunctions on effect propositions. Then, the above domain can be described by the following propositions:

[1]Vladimir Lifschitz's email message to lmp@di.fct.unl.pt and renwei@di.fct.unl.pt on March 25, 1996.

$$
\begin{array}{l}
\{[toggle(left)\ \textbf{causes}\ \ on(big)\ \textbf{if}\ \ \neg on(big)] \\
\wedge[toggle(left)\ \textbf{causes}\ \ \neg on(big)\ \textbf{if}\ \ on(big))]\} \\
\dot{\vee} \\
\{[toggle(left)\ \textbf{causes}\ \ on(small)\ \textbf{if}\ \ \neg on(small)] \\
\wedge[toggle(left)\ \textbf{causes}\ \ \neg on(small)\ \textbf{if}\ \ on(small))]\}
\end{array}
$$

$$
\begin{array}{l}
\{[toggle(right)\ \textbf{causes}\ \ on(big)\ \textbf{if}\ \ \neg on(big)] \\
\wedge[toggle(right)\ \textbf{causes}\ \ \neg on(big)\ \textbf{if}\ \ on(big))]\} \\
\dot{\vee} \\
\{[toggle(right)\ \textbf{causes}\ \ on(small)\ \textbf{if}\ \ \neg on(small)] \\
\wedge[toggle(right)\ \textbf{causes}\ \ \neg on(small)\ \textbf{if}\ \ on(small))]\}
\end{array}
$$

$$
\begin{array}{l}
\{[toggle(left)\ \textbf{causes}\ \ on(big)\ \textbf{if}\ \ \neg on(big)] \\
\wedge[toggle(left)\ \textbf{causes}\ \ \neg on(big)\ \textbf{if}\ \ on(big))]\} \\
\dot{\vee} \\
\{[toggle(right)\ \textbf{causes}\ \ on(big)\ \textbf{if}\ \ \neg on(big)] \\
\wedge[toggle(right)\ \textbf{causes}\ \ \neg on(big)\ \textbf{if}\ \ on(big))]\}
\end{array}
$$

$$
\begin{array}{l}
\{[toggle(left)\ \textbf{causes}\ \ on(small)\ \textbf{if}\ \ \neg on(small)] \\
\wedge[toggle(left)\ \textbf{causes}\ \ \neg on(small)\ \textbf{if}\ \ on(small))]\} \\
\dot{\vee} \\
\{[toggle(right)\ \textbf{causes}\ \ on(small)\ \textbf{if}\ \ \neg on(small)] \\
\wedge[toggle(right)\ \textbf{causes}\ \ \neg on(small)\ \textbf{if}\ \ on(small))]\}
\end{array}
$$

It can be seen that finite uncertainties have been represented by exclusive disjunction $\dot{\vee}$. Intuitively, one of the following two domain descriptions should be real.

$$
\begin{array}{l}
toggle(left)\ \textbf{causes}\ \ on(small)\ \textbf{if}\ \ \neg on(small) \\
toggle(left)\ \textbf{causes}\ \ \neg on(small)\ \textbf{if}\ \ on(small) \\
toggle(right)\ \textbf{causes}\ \ on(big)\ \textbf{if}\ \ \neg on(big) \\
toggle(right)\ \textbf{causes}\ \ \neg on(big)\ \textbf{if}\ \ on(big)
\end{array}
$$

and

$$
\begin{array}{l}
toggle(left)\ \textbf{causes}\ \ on(big)\ \textbf{if}\ \ \neg on(big) \\
toggle(left)\ \textbf{causes}\ \ \neg on(big)\ \textbf{if}\ \ on(big) \\
toggle(right)\ \textbf{causes}\ \ on(small)\ \textbf{if}\ \ \neg on(small) \\
toggle(right)\ \textbf{causes}\ \ \neg on(small)\ \textbf{if}\ \ on(small)
\end{array}
$$

Later we will see that our methodology works well and produces what is intuitively acceptable. The rest of the paper is organized as follows. In Section 2 we present an action description language, denoted $\mathcal{A}^+$, which is an extension to $\mathcal{A}$. The reason we choose $\mathcal{A}$ is simply that $\mathcal{A}$ has been shown to be a simple, extensible and expressive action description language, and to be equivalent to other three major formalisms [8] proposed by Pednault

[16], Reiter [18] and Baker [2], respectively. In Section 3 we will present a translation from domain descriptions in $\mathcal{A}^+$ to abductive logic programs. This translation will serve to bridge the reasoning about actions and abductive logic programming. Generally it is not easy or simple to refine action theories or to predict and explain in $\mathcal{A}^+$. The translation will effectively reduce working in $\mathcal{A}^+$ to working in an abductive logic programming system, thereby being automated. In Section 4 we will show that our translation is both sound and complete. In Section 5 we will discuss tests and refinements by using abductive logic programming. In Section 6 we return to Lifschitz' challenge problem. In Section 7 we conclude this paper with a few remarks. For the space limit we will omit all proofs of technical results, but can be found in the extended version [13] of this paper.

2 Domain Descriptions

In this section we present an action description language $\mathcal{A}^+$, an extension to $\mathcal{A}$ of [6].

2.1 Syntax

We begin with three disjoint non-empty sets of symbols, called *proposition names*, *fluent names*, and *action names*, respectively. For convenience we will also use parameterized names. *Actions* and *propositions* are defined to be action names and proposition names, respectively. A *fluent expression*, or simply *fluent*, is defined to be a fluent name possibly preceded by $\neg$. A fluent expression is also called a *positive fluent* if it only consists of a fluent name; otherwise it is called a *negative fluent*.

In $\mathcal{A}^+$, a domain description is defined to be a set of effect assertions and constraints. An effect assertion is defined to be a statement of the form

$$A \textbf{ causes } F \textbf{ if } P_1, \ldots, P_m, Q_1, \ldots, Q_n$$

where A is an action, each of F, P_1, ..., P_m $(m \geq 0)$ is a fluent expression, and each of $Q_1, \ldots, Q_m$ $(n \geq 0)$ is a proposition name. If $m = n = 0$, then we will simply write it as A **causes** F. A constraint is defined as follows:

- A proposition name is an atomic constraint.
- A statement of the form

  $$F \textbf{ after } A_1, \ldots, A_n$$

  where F is a fluent and A_i is an action, is an atomic constraint, also called *value assertion*. When $n = 0$, the value assertion above is abbreviated to **initially** F.

- If C_1 and C_2 are constraints, then $\neg C_1$, $C_1 \wedge C_2$, $C_1 \vee C_2$ are constraints, called complex constraints. Other propositional connectives can be defined in terms of them as derived connectives.

It can be seen that $\mathcal{A}^+$ is an extension of $\mathcal{A}$ by allowing propositions and more types of constraints. However, the detailed discussion on relations between $\mathcal{A}$ and $\mathcal{A}^+$ is out of this paper.

2.2 Remarks

It seems that we would increase the expressive power if we defined the effect assertions in the following way: (1) A basic effect assertion is a statement of the form A **causes** F **if** $C_1, \ldots, C_n$; (2) An effect assertion is a statement of the form $(E_{11} \wedge \ldots \wedge E_{1m_1}) \vee \ldots \vee (E_{n1} \wedge \ldots \wedge E_{nm_n})$, where each E_{ij} is a basic effect assertion. In fact, combining with proposition names, we can reduce the above complex effect assertion to simpler ones of $\mathcal{A}^+$. We can systematically do so by introducing a few new proposition names and then transform effect assertions. For example, consider:

$$\begin{array}{l} (A_1 \textbf{ causes } F_1 \textbf{ if } C_{11}, \ldots, C_{1n_1}) \\ \vee \ldots \\ \vee (A_m \textbf{ causes } F_m \textbf{ if } C_{m1}, \ldots, C_{mn}) \end{array}$$

Let h_i, $1 \le i \le m$ be m new proposition symbols. Then, the above complex effect assertions can be transformed into m basic effect assertions and a new constraint as follows:

$$\begin{array}{l} A_1 \textbf{ causes } F_1 \textbf{ if } C_{11}, \ldots, C_{1n_1}, h_1 \\ \ldots \\ A_m \textbf{ causes } F_m \textbf{ if } C_{m1}, \ldots, C_{mn}, h_m \\ h_1 \vee \ldots \vee h_m \end{array}$$

On the other hand, it also seems that we would increase the expressive power if we allowed general well-formed propositional formulas in the preconditions of effect assertions. For example, let A be an action, P_1 a fluent, and Q_1, Q_2, Q_3 be proposition names. Consider

$$A \textbf{ causes } F \textbf{ if } P_1, (Q_1 \wedge Q_2) \vee \neg Q_3$$

This kind of seemingly more expressive effect assertions can also be reduced to effect assertions in $\mathcal{A}^+$. Let Q_4 be a new proposition name. The following effect assertion and a constraint is equivalent to the above assertion:

$$\begin{array}{l} A \textbf{ causes } F \textbf{ if } P_1, Q_4 \\ Q_4 \leftrightarrow (Q_1 \wedge Q_2) \vee \neg Q_3 \end{array}$$

2.3 Semantics

The semantics of a domain description is defined by using proposition assignment, states, and transitions.

A proposition assignment α is a set of proposition names. Given a proposition name P and an assignment α, we say that P is true if $P \in \alpha$, and $\neg P$ is true if $P \notin \alpha$. A *state* is a set of fluent names. Given a fluent name F and a state σ, we say that F holds in σ if $F \in \sigma$; $\neg F$ holds in σ if $F \notin \sigma$. A *transition function* Φ is a mapping from the set of pairs (A, σ), where A is an action expression and σ is a state, to the set of states.

An *interpretation structure* is a triple (α, σ_0, Φ), where α is an assignment, σ_0 is a state, called the *initial state* of (σ_0, Φ), and Φ is a transition function. For any interpretation structure $M = (\alpha, \sigma_0, \Phi)$ and any sequence of action expressions $A_1; \ldots; A_m$ in M, by $\Phi(A_1; \ldots; A_m, \sigma_0)$ we denote the state $\Phi(A_m, \Phi(A_{m-1}, \ldots, \Phi(A_1, \sigma_0) \ldots))$.

Given an interpretation structure (α, σ_0, Φ), a constraint C is said to be true with respect to it iff

- if C is a proposition name, then $C \in \alpha$;
- if C is a value assertion of the form F **after** $A_1, \ldots, A_n$, then F holds in the state $\Phi(A_1; \ldots; A_n, \sigma_0)$;
- if C is a complex constraint, then it is true according to the usual propositional connective evaluation methods.

An interpretation structure (α, σ_0, Φ) is a *model* of a domain description D iff

- Every constraint is true with respect to the interpretation structure.
- For every action A, every fluent name F, and every state σ: (i) If D includes an effect assertion A **causes** F **if** $P_1, \ldots, P_m, Q_1, \ldots, Q_n$, such that fluents $P_1, \ldots, P_m$ hold in σ and propositions $Q_1, \ldots, Q_n$ are true with respect to (α, σ_0, Φ), then $F \in \Phi(A, \sigma)$; (ii) If D includes an effect assertion A **causes** $\neg F$ **if** $P_1, \ldots, P_m, Q_1, \ldots, Q_n$, such that fluents $P_1, \ldots, P_n$ hold in σ and propositions $Q_1, \ldots, Q_n$ are true with respect to (α, σ_0, Φ), then $F \notin \Phi(A, \sigma)$; (iii) If D does not include any such effect assertions, then $F \in \Phi(A, \sigma)$ iff $F \in \sigma$.

A domain description is *consistent* if it has a model. A domain description is *complete* if it has exactly one model. A domain description D entails a value assertion V if V is true in all models of D. It can be shown that different models of the same domain description differ only in different initial states and/or proposition assignments. In addition, the interpretation of a proposition name is independent of states.

In reality a practical domain should have only one model. The task of refining domain descriptions is to construct a new domain description which has fewer models than the original domain description. We will achieve this purpose by first performing some actions and observing their outcome, then we will abductively infer the truth values of propositions and initial states. We will make use of abductive logic programming for the purpose of abductive reasoning.

3 Translation into Abductive Programs

In this section we will present a translation from domain descriptions into abductive logic programs. An abductive logic program is a triple $< P, IC, \Delta >$, where P is a set of logic programming rules, IC is a set of first-order sentences as constraints, and Δ is a set of predicates, called abducible predicates. An abductive answer δ to a query Q in $< P, IC, \Delta >$ is a finite subset of ground instances of Δ such that (i) $Q \in SEM(P \cup \{a \leftarrow \ : a \in \delta\}, IC)$; (ii) $P \cup \{a \leftarrow \ : a \in \delta\}, IC)$ is consistent according to definition of SEM; (iii) δ is minimal in the sense that no subset of it satisfies the previous two conditions, where $SEM(P, IC)$ denotes the semantics of the program P with constraints IC. There have been a few competing semantics in the literature: predicate completion semantics, stable model semantics, and well-founded model semantics. Later we will see that our logic program translations are acyclic, and thus all of these major semantics agree. Therefore we will define the semantics of logic programs as the predicate completion semantics. For abductive logic programs, we will complete all predicates except the abducible ones [3].

Let D be a domain description. The translation πD includes a set of programming rules and a set of constraints defined as follows:

1. Initialization: $holds(F, s_0) \leftarrow initially(F)$.

2. Law of Inertia:

$$holds(F, result(A, S)) \leftarrow holds(F, S), not\ noninertial(F, S, A).$$

   where not is the negation-as-failure operator. By the law of inertia, F is true at a new situation by doing A on S if it was true at S.

3. Each effect assertion a **causes** f **if** $p_1, \ldots, p_m, q_1, \ldots, q_n$, with f being positive, p_i being a fluent, and q_i being a proposition, is translated into

$$holds(f, result(a, S)) \leftarrow holds(p_1, S), \ldots, holds(p_m, S), q_1, \ldots, q_n.$$

   where $holds(\neg p, S)$ with p being positive stands for $not\ holds(p, S)$. This convention is also used in the rest of this paper.

4. Each effect assertion a **causes** $\neg f$ **if** $p_1, \ldots, p_m, q_1, \ldots, q_n$, with f being positive, p_i being a fluent, and q_i being a proposition, is translated into

$$noninertial(f, a, S) \leftarrow holds(p_1, T), \ldots, holds(p_m, T), q_1, \ldots, q_n.$$

5. For every constraint C of D: (i) if C is a proposition name, $\pi C = C$; (ii) if C is f **after** $a_1, \ldots, a_n$ with f being positive, then $\pi C = holds(f, result(a_1; \ldots; a_n, s_0))$; (iii) if C is $\neg f$ **after** $a_1, \ldots, a_n$, with f being positive, then $\pi C = \neg holds(f, result(a_1; \ldots; a_n, s_0))$; (iv) $\pi(\neg C_1) = \neg(\pi C_1)$, $\pi(C_1 \wedge C_2) = \pi C_1 \wedge \pi C_2$, $\pi(C_1 \vee C_2) = \pi C_1 \vee \pi C_2$.

We will define abducible predicates to be $initially(F)$ and all proposition names. The semantics of πD, denoted by $Comp(\pi D)$, is defined to be the first-order theory by completing all predicates except $initially(F)$ and proposition names, jointly with Clark's theory of equality, and the constraints [3].

Theorem 3.1 *Let D be any domain description in $\mathcal{A}^+$. πD is an acyclic logic program with first-order constraints in the sense of [1].*

Corollary 3.2 *The completion semantics $Comp(\pi D)$ of πD agrees with its generalized stable model semantics [7] and generalized well-founded model semantics [17].*

The above corollary means that the result of this paper can be experimented with any abductive logic programming system with one of the three major semantics. The detailed proof follows from [5].

4 Soundness and Completeness

In general it is very difficult to reason about actions in $\mathcal{A}^+$. The purpose of the translation is to reduce the reasoning work in $\mathcal{A}^+$ to abductive querying in an abductive logic programming system. This section will show that reasoning in $\mathcal{A}^+$ is equivalent to abductive querying.

Theorem 4.1 *The translation π is sound. That is, for any domain description D and any value assertion V, if $Comp(\pi D) \models \pi V$, then D entails V.*

Definition 4.2 *A domain description D is effect consistent iff for each pair of effect assertions,*

$$A \textbf{ causes } F \textbf{ if } C_1, \ldots, C_m$$
$$A \textbf{ causes } \neg F \textbf{ if } C_{m+1}, \ldots, C_n$$

in D, there exists i, $1 \leq i \leq m$, and j, $m+1 \leq j \leq n$, such that C_i is the complement of C_j.

Theorem 4.3 *The translation π is complete for any effect consistent domain descriptions. That is, for any effect consistent domain description D and any value assertion V, if D entails V, then $Comp(\pi D) \models \pi V$.*

The requirement for a domain description to be effect consistent is necessary. If a domain description D is not effect consistent, no transition functions exist to satisfy its effect assertions, thus it has no models, and hence it entails every value assertion. On the other hand, its translation is consistent and thus has at least one model which entails a proper subset of what D entails.

The above soundness and completeness theorems signify that our translation can actually be used for the general purposes of reasoning about actions and changes such as abductive planning, prediction, explanation. That is to say, our result of this paper goes beyond refinement of action theories. But we will not delve into detailed discussion on how to use our translation for abductive planning, temporal prediction and explanation. In the next section we will concentrate on refinement of action theories.

5 Refinement

Let D be a domain description. D may have more than one model. If D has more than one model, we may only predict a disjunctive future instead of a definite future. That is to say, after a sequence of actions is done, we cannot predict whether a fluent is definitely true or not. When a domain description is complete, we can always predict whether a fluent is true or not after an action is done. This is sometimes a very important factor in reasoning about actions, as shown as in [14].

When a domain description is not complete, all its models differ in their initial states and/or proposition assignments. In order to determine initial states and proposition assignments, one may perform some tests: doing some actions, observing their effects, and then abductively determining initial states and proposition names.

Now suppose that we are given a domain description D_0. We want to refine it. The way to do it, as said as before, is to perform some actions and observe their effects. This process is called *test*. The purpose of tests is to generate new value assertions. And thus we can formally define a test to be a set of value assertions.

Definition 5.1 *A* test *τ in an action domain is a set of value assertions. Let D be a domain description. The pair (D, τ) is called a refinement problem.*

Theorem 5.2 *Let D be a domain description, and τ a test. Then, every model of $D \cup \tau$ is a model of D.*

Note that the converse of the above theorem does not hold in general cases. The above theorem means that simply adding tests to a domain description will definitely give a better and new domain description. But syntactically $D \cup \tau$ is more complicated than D. We may prefer simpler and finer descriptions. Note that in an interpretation structure, all proposition names will be either true or false. In the reality, all these proposition names can and can only be either true or false. When we do enough tests, the refinement of the domain will be closer and closer to a complete domain description. This implies that the complete domain description is a limit of all refinements of domain descriptions. When the domain description has only one model, all proposition names can be removed from the domain description by substituting them with their truth values, and thus syntactically simplifying the domain description. Hence, we have the following definition of refinements:

Definition 5.3 *Let D_1 and D_2 be two domain descriptions. D_2 is said to be a refinement of D_1 iff the following conditions are satisfied:*

- *Every model of D_2 is a model of D_1;*
- *There is no proposition name in D_2 which is true in every model of D_2;*
- *There is no proposition name in D_2 which is false in every model of D_2.*

In what follows we want to show how to compute refinements with abductive logic programming. In Section 3 we presented a translation from domain descriptions to abductive logic programs. However, many existing abductive logic programming systems do not directly support our constraints. Instead, they support constraints of the form

$$\bot \leftarrow L_1, \ldots, L_n$$

First we need to translate all constraints into the above form.

The translation, still denoted by π, is as follows. Let C be a constraint in the program πD. Then C can be equivalently transformed into a conjunctive normal form:

$$(C_{11} \vee \ldots \vee C_{1m_1}) \wedge \ldots \wedge (C_{m1} \vee \ldots \vee C_{mn})$$

Then, it will be translated into

$$\begin{array}{lll} \bot & \leftarrow & not\ C_{11}, \ldots, not\ C_{1m_1} \\ & \ldots & \\ \bot & \leftarrow & not\ C_{m1}, \ldots, not\ C_{mn} \end{array}$$

where *not* $\neg L$ is taken as L.

After constraints are translated into a logic program, we can run it in any abductive logic programming system. Before proceeding, we need to guarantee that the correctness of the translation is preserved.

Theorem 5.4 *The translation π is both sound and complete for any effect consistent domain descriptions.*

Note that above translation does not change completions of non-abducible predicates. On the other hand, the completion of $\perp$ is logically equivalent to $\neg C$. Therefore, the semantics of new programs is the same as before.

Let $\tau = \{V_1, \ldots, V_n\}$ be a test. Then, τ can be transformed into a query:

$$\leftarrow \pi V_1, \ldots, \pi V_n$$

where for each i, πV_i is defined as follows: Let V_i be F **after** $A_1, \ldots, A_n$ in τ. If F is positive, then πV_i is defined to be $holds(F, result(A_1; \ldots; A_n, s_0))$; if F is negative and equal to $\neg G$, then πV_i is defined to be $not\ holds(G, result(A_1; \ldots; A_n, s_0))$.

Submitting the query to an abductive logic programming system, we will get abductive answers to it. In what follows we will write $\mathcal{R}(D, \tau)$ to stand for the set of all abductive answers to the query $\leftarrow \pi\tau$ against the abductive logic program πD. Now we are in a position to define the procedure of refining action theories.

Definition 5.5 *Let D be a domain description and τ a test. Let $\mathcal{R}(D, \tau) = \{R_1, \ldots, R_n\}$. Perform:*

1. *For every proposition name P, if $P \notin R_1 \cup \ldots \cup R_n$, remove from D all effect assertions containing P in the precondition list, and replace P with $false$ in every constraint of D;*

2. *For every proposition name P, if $P \in R_1 \cap \ldots \cap R_n$, remove P from all effect assertions of D, and replace P with true in every constraint of D;*

3. *Simplify constraints of D in the usual way by using of $true$ and $false$ in the formulas. For example, if C is of the form $\neg false$ or $C_1 \vee true$, C is removed.*

Then, Define $\mathcal{S}(D, \tau)$ to be the set of the resulting effect assertions, constraints, and the test τ.

The following theorem says that the new domain description $\mathcal{S}(D, \tau)$ is a refinement of D.

Theorem 5.6 *Let D be a domain description, τ a test. Then, $\mathcal{S}(D, \tau)$ is a refinement of D.*

6 An Example

Now we return to the example in the Introduction. Let $controls(S, L)$ be a parameterized proposition name to denote that switch S controls light L. Then, we can have the following domain description D:

$controls(left, small) \leftrightarrow controls(right, big)$
$controls(left, big) \leftrightarrow controls(right, small)$
$controls(left, small) \dot{\vee} controls(left, big)$
$controls(right, small) \dot{\vee} controls(right, big)$
$toggle(left)$ **causes** $on(small)$ **if** $\neg on(small), controls(left, small)$
$toggle(left)$ **causes** $\neg on(small)$ **if** $on(small), controls(left, small)$
$toggle(right)$ **causes** $on(small)$ **if** $\neg on(small), controls(right, small)$
$toggle(right)$ **causes** $on(small)$ **if** $\neg on(small), controls(right, small)$
$toggle(left)$ **causes** $on(big)$ **if** $\neg on(big), controls(left, big)$
$toggle(left)$ **causes** $\neg on(big)$ **if** $on(big), controls(left, big)$
$toggle(right)$ **causes** $on(big)$ **if** $\neg on(big), controls(right, big)$
$toggle(right)$ **causes** $on(big)$ **if** $\neg on(big), controls(right, big)$
initially $\neg on(big)$
initially $\neg on(small)$

Then, we have an abductive logic program πD. Now suppose we have a test $\tau = \{on(big)$ **after** $toggle(left)\}$. Then, evaluate the following query against the abductive logic program πD:

$$\leftarrow holds(on(big), result(toggle(left), s_0))$$

An abductive logic programming system should correctly output the following unique abductive answer $\mathcal{R}(D, \tau)$:

$$\{\{controls(right, small), controls(left, big)\}\}$$

Then, by definition we have the following new domain description $\mathcal{S}(D, \tau)$:

$toggle(right)$ **causes** $on(small)$ **if** $\neg on(small)$
$toggle(right)$ **causes** $on(small)$ **if** $\neg on(small)$
$toggle(left)$ **causes** $on(big)$ **if** $\neg on(big)$
$toggle(left)$ **causes** $\neg on(big)$ **if** $on(big)$
initially $\neg on(big)$
initially $\neg on(small)$
$on(big)$ **after** $toggle(left)$

Thus we have obtained a complete domain description which enables us to generate plan, to predict the future, or to explain the past, as what we expected and intended.

7 Concluding Remarks

In this paper we have presented an experiment on using the abductive logic programming paradigm to refine an action theory in line with [10, 11] starting from [6]. An action theory, also called domain description, describes effects of actions and initial states in a dynamic domain. A complete action theory should enable us to determine which fluent will be true and which fluent will be false after an action is performed. A complete action theory can be used for planning, prediction and explanation. In practice we may encounter incomplete domains with finite uncertainties. The finite uncertainties may be removed by doing some tests and abductive reasoning. Technically we presented an action description language $\mathcal{A}^+$ for domain descriptions, then we presented a translation from $\mathcal{A}^+$ to abductive logic programs. The translation has been shown to be both sound and complete. Thus, the task of reasoning about actions in $\mathcal{A}^+$ amounts to abductive query evaluation in abductive logic programming systems. We also indicate that our abductive logic program is acyclic, and thus we can use any abductive query evaluation procedure, no matter whether their semantics is based on predicate completion, stable models, or well-founded models. The test on a domain is a set of observed effects of a sequence of specific actions. The test can be used to determine truth values of proposition names which serve to represent uncertainties. This has been tested with the latest version of a meta-interpreter of abductive logic programs [4]. To the best of our knowledge, there is no similar work in this topic, although there have been many reports on $\mathcal{A}$ family languages. In general, the refinement of action theories can be regarded as learning. But this kind of learning is different from the main-trend work on learning, where generalization, specialization, and induction is often used as the inference mechanism. In this paper we have used abduction as the underlying inference mechanism. The result of this paper is currently used to develop intelligent situated agent [12], which is able to observe, act and reason in the real world.

Acknowledgement

This work was partially supported by JNICT of Portugal under PRAXIS 2/2.1/TIT/1593/95 and PRAXIS XXI/BPD/4165/94 and NSERC of Canada under 31-611024.

References

[1] K. R. Apt and M. Bezem. Acyclic programs. In *Proc. of ICLP 90*, pages 579–597. MIT Press, 1990.

[2] A. B. Baker. Nonmonotonic reasoning in the framework of situation calculus. *Artificial Intelligence*, 49:5–23, 1991.

[3] L. Console, D. T. Dupre, and P. Torasso. On the relationship between abduction and deduction. *Journal of Logic and Computation*, 1(5):661–690, 1991.

[4] C. V. Damásio, L.M. Pereira, and W. Nejdle. Revise: An extended logic programming system for revising knowledge bases. In *Proc. of KR'94*, 1994.

[5] M. Denecker. Knowledge representation and reasoning in incomplete logic programming. Ph.D. thesis, Department of Computer Science, K.U.Leuven, 1993.

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

[7] A.C. Kakas and P. Mancarella. Generalized stable models: A semantics for abduction. In *Proc. of ECAI'90*, 1990.

[8] G.N. Kartha. Soundness and completeness theorems for three formalizations of action. In *Proc. IJCAI93*, pages 712–718. MIT Press, 1993.

[9] R.A. Kowalski and F. Sadri. The situation calculus and event calculus compared. In *Proc. of ILPS 94*, pages 539–553. MIT Press, 1994.

[10] R. Li and L.M. Pereira. Temporal reasoning with abductive logic programming. In W. Wahsler, editor, *Proc. of ECAI'96*, pages 13–17. John Wiley & Sons, 1996.

[11] R. Li and L.M. Pereira. What is believed is what is explained (sometimes). In *Proc. of AAAI'96*, pages 550–555, 1996.

[12] R. Li and L.M. Pereira. Knowledge-based situated agents among us. In J. P. Muller, M. J. Wooldridge, and N. R. Jennings, editors, *Intelligent Agents III – Proc. of the Third International Workshop on Agent Theories, Architectures, and Languages (ATAL-96), LNAI 1193*, pages 375–389. Springer, 1997.

[13] R. Li, L.M. Pereira, and V. Dahl. Refining action theories through abductive logic programming (extended version). In *Manuscript*, 1997.

[14] F. Lin and Y. Shoham. Provably correct theories of actions: preliminary report. In *Proc. of AAAI-91*, 1991.

[15] J. McCarthy and P.J. Hayes. Some philosophical problems from the stand-point of artificial intelligence. In B. Meltzer and D. Michie, editors, *Machine Intelligence*, volume 4, pages 463–502, Edinburgh, 1969.

[16] E. P. D. Pednault. Adl: Exploring the middle ground between strips and the situation calculus. In R. J. Brachman, H. Levesque, and R. Reiter, editors, *Proc. of KR'89*, pages 324–332. Morgan Kaufmann Publishers, Inc., 1989.

[17] L. M. Pereira, J. J. Alferes, and J. N. Aparício. Nonmonotonic reasoning with well founded semantics. In K. Furukawa, editor, *Proc. of 8th ICLP*, pages 475–489. MIT Press, 1991.

[18] 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 359–380. Academic Press, San Diego, CA, 1991.

[19] E. Sandewall. *Features and Fluents: The Representation of Knowledge about Dynamic Systems, Vol. 1.* Oxford University Press, 1994.