Temporal Reasoning with Abductive Logic Programming

Renwei Li and **Luís Moniz Pereira**[1]

Abstract In this paper we extend Gelfond and Lifschitz' action description language $\mathcal{A}$ with concurrent actions and give a new semantics to concurrent actions. In order to automate temporal reasoing we present a translation from domain descriptions in the resulting new action description language to abductive logic programs. The translation has been shown to be both sound and complete. Our method is applicable to the temporal projection problem with incomplete information, as well as to reasoning about the past.

1 INTRODUCTION

In [11] Gelfond and Lifschitz proposed an action description language $\mathcal{A}$ and presented a translation from $\mathcal{A}$ to extended logic programs with answer sets as semantics [10], and proved its soundness. Kartha proposed three sound and complete translations of $\mathcal{A}$ into three formalisms of reasoning about actions proposed by Baker [2], Pednault [20], and Reiter [22], respectively. Kartha and Lifschitz [16] also considered how to deal with ramifications based on $\mathcal{A}$ by using nested abnormality theory [17]. Dung presented a translation of domain descriptions in $\mathcal{A}$ into abductive logic programs with database updates as applications [8]. Denecker and de Schreye proposed another much simpler translation of $\mathcal{A}$ into abductive normal logic programs with constraints [7] and showed that their translation is both sound and complete with respect to the predicate completion semantics [4] of abductive normal logic programs. Baral and Gelfond [3] extended the action language $\mathcal{A}$ with concurrent actions. All these results impress us that the action description language $\mathcal{A}$ is a good basis for systematical development of theories of actions.

This paper is in line with the work related to the action description language $\mathcal{A}$. We will extend $\mathcal{A}$ with concurrent actions in a way different from [3], and present a translation from domain descriptions in the new action description language to abductive normal logic programs. The translation has been shown to be both sound and complete for any domain descriptions. Then temporal reasoning is reduced to abductive query evaluation in abductive logic programs. Our method is applicable to the temporal projection problem with incomplete information, and can also be applied to temporal explanation of inferring actions from fluent changes.

The rest of this paper is organized as follows. In Section 2 we informally discuss concurrent actions and their semantics. In Section 3 we extend $\mathcal{A}$ with concurrent actions. In Section 4 we present a sound and complete translation from domain descriptions in the new action language to abductive logic programs. In Section 5 we compare our work with others. Finally, in Section 6, we draw some conclusions.

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

ECAI 96. *12th European Conference on Artificial Intelligence*
Edited by W. Wahlster
Published in 1996 by John Wiley & Sons, Ltd.

2 CONCURRENT ACTIONS

In reality, occurrences of many actions overlap in time, which complicates temporal prediction and explanation in AI. For definite goals some actions may be planned to be carried out at the same time in order to save time, or to decrease production cost, or for other context-dependent purposes.

A concurrent action is understood as a finite nonempty set of atomic actions, which happen at the same time. A nonempty subset of a concurrent action is called a subaction of the concurrent action. For example, opening a door and switching on a light at the same time is a concurrent action consisting of two atomic actions: opening a door and switching on a light. In common sense, the effect of a concurrent action is the aggregation of those of its subactions. For example, after one opens a door and switches on a light at the same time, the door is open and the light is on, which is the aggregation of the effects of opening the door and switching on the light.

The effect relationship between an action and its subactions was first discussed by Gelfond, Lifschitz and Robinov in the framework of the situation calculus [12], and further explored by Lin and Shoham [19], and Baral and Gelfond [3].

In [19], Lin and Shoham identified a problem called the *action-oriented frame problem* and called the traditional frame problem the *fluent-oriented frame problem*. Their work was motivated by their observation on the similarity and symmetrical roles played by actions and situations. Moreover, Lin and Shoham provided a solution to the action-oriented frame problem and proved its correctness with respect to a formal adequacy criterion called *epistemological completeness*, proposed in [18] by themselves. Intuitively, a theory of an action is epistemologically complete if, given a complete description of the initial situation, the theory enables us to predict a complete description of the resulting situation when the action is performed.

In [3], Baral and Gelfond extended $\mathcal{A}$ with concurrent actions and obtained an action language, denoted by $\mathcal{A}_{\mathcal{BG}}$ in this paper. When a domain is described in $\mathcal{A}_{\mathcal{BG}}$ the following postulate is adopted and employed: A concurrent action *usually* inherits effects of its subactions.This is essentially the same as that of [19] . But their techniques are different. Baral and Gelfond used the negation-as-failure operator to formalize the inheritance postulate in extended logic programming with answer-set semantics [10], while Lin and Shoham used circumscription policy to deal with the action-oriented frame problem. Another difference, more important, between [19] and [3] is in their treatment of conflicting effects, as analyzed as follows:

Let $Open$ and $Close$ to denote two atomic actions: to make sure to open and close the door, respectively. In common sense, $Open$ and $Close$ have conflicting effects. Now consider a concurrent action $\{Open, Close\}$. How should we evaluate and deal with concurrent actions with conflicting effects? Lin-Shoham and Baral-Gelfond give two different methods.

In Lin and Shoham's solution, it is required to be able to derive whether or not the door is closed in the next situation in order to guarantee the epistemological completeness. The epistemological completeness implies that in any situation one can always decide whether a fluent is true or not. Thus, if the door is initially closed, then the door will still be closed after $\{Open, Close\}$ is done; if the door is initially open, then the door will still be open after $\{Open, Close\}$ is done. This is true when both actions exert exactly the same force according to mechanics. But it is not the case in common sense, and it does not seem to be well justified that we could predict a complete description of the resulting situation after the concurrent action $\{Open, Close\}$ is done.

In Baral and Gelfond's solution, after $\{Open, Close\}$ is done, the resulting situation does not exist and $\{Open, Close\}$ is not executable. Baral and Gelfond define the state (situation) transition by doing actions as a partial mapping. For those concurrent actions with conflicting effects such as $\{Open, Close\}$, the state transition is not defined. By [3], any successive actions are not executable after $\{Open, Close\}$ is done. Now assume we have an additional action, $Switch$, which denotes switching a light. According to [3] the action sequence $\{Open, Close\}; Switch$ is unexecutable. In common sense, the status of the light should change, although the status of the door is not known. Now consider another concurrent action $\{Open, Close, Switch\}$. According to [3] the concurrent action $\{Open, Close, Switch\}$ is not executable, and the resulting situation is not defined. Although $Open$ and $Close$ have confliciting effects, we can at least infer that the status of the light will be changed after $\{Open, Close, Switch\}$ is done.

In this paper we will give a new semantics to concurrent actions. When subactions of a concurrent action do not have conflicting effects, it can be proved that our semantics coincides with Lin-Shoham's and Baral-Gelfond's. When two subactions of a concurrent action have conflicting effects, the truth value of all fluents affected by the two subactions will be left undefined. For example, the fluent denoting the status of the door will be left undefined after $\{Open, Close\}$ is done. In our new semantics, we can infer, for example, the door will be closed after the sequence of actions $\{Open, Close\}; Close$ is done, the status of the light will be changed after $\{Open, Close, Switch\}$ or $\{Open, Close\}; Switch$ is done. We think that our new semantics is closer to our common-sense reality than the semantics of [19] and [3]. The details of the semantics will be given in Subsection 3.2. Reasonign about effects of (concurrent) actions is automated with abductive logic programming, and temporal reasoning is thus reduced to abductive query evaluation.

3 AN ACTION DESCRIPTION LANGUAGE AND ITS SEMANTICS

In this section we extend Gelfond and Lifschitz' action language $\mathcal{A}$ with concurrent actions, and give a new semantics to concurrent actions. The resulting language is denoted by $\mathcal{AC}$.

3.1 Syntax of $\mathcal{AC}$

We begin with the alphabet Σ of a domain, which consists of two disjoint non-empty sets of symbols Σ_f and Σ_a, called *fluent names* and *atomic action names*, respectively. We will often use self-explanatory fluent names and atomic action names when no confusion arises.

A *fluent expression* 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*.

An *action expression* is defined to be a non-empty finite set of atomic action names. We say that an action expression A is a *subaction expression* (*proper subaction expression*) of B iff $A \subseteq B$ ($A \subset B$). If A is a subaction expression of B, we often say that B contains A. We will simply write A for $\{A\}$ if A is atomic. An action expression denotes a concurrent action composed of atomic actions which happen at the same time. Fluent expressions and action expressions will simply be called fluents and actions, respectively, if no confusion arises.

In $\mathcal{AC}$ there are two kinds of propositions: value propositions and effect propositions, simply called v-propositions and e-propositions. A v-proposition is a statement of the form

$$F \textbf{ after } A_1; \ldots; A_m \qquad (1)$$

where F is a fluent expression and $A_1, \ldots, A_m (m \geq 0)$ are action expressions. If $m = 0$, then we will write it as **initially** F. An e-proposition is a statement of the form

$$A \textbf{ causes } F \textbf{ if } P_1, \ldots, P_n \qquad (2)$$

where A is an action expression, and each of $F, P_1, \ldots, P_n (n \geq 0)$ is a fluent expression. If $n = 0$, then we will write it as A **causes** F.

Given the alphabet of a domain, its description D is a set of v-propositions and e-propositions.

For example, a door-light domain is described as follows: D_{door} = \{ **initially** $\neg Closed.$ **initially** $On. Open$ **causes** $\neg Closed. Close$ **causes** $Closed.$ $Switch$ **causes** On **if** $\neg On.$ $Switch$ **causes** $\neg On$ **if** On\}. The Yale Shooting domain [13] is as follows: D_{YSP} = \{ **initially** $\neg Loaded.$ **initially** $Alive. Shoot$ **causes** $\neg Alive$ **if** $Loaded.$ $Shoot$ **causes** $\neg Loaded.$ $Load$ **causes** $Loaded$\}. The Murder Mystery domain is obtained from D_{YSP} by substituting $\neg Alive$ **after** $Shoot; Wait$ for **initially** $\neg Loaded$. Mary's Soup [12] domain is defined as follows. Whenever Mary lifts the bowl with only one hand, she will spill the soup. But when she lifts the bowl with two hands, she will not spill the soup. Let LL and LR to denote lifting the bowl with the left hand and the right hand, respectively. Then, the domain can be described as follows: D_{soup} = \{ **initially** $\neg Spilled.$ LL **causes** $Spilled.$ LR **causes** $Spilled.$ $\{LL, LR\}$ **causes** $\neg Spilled$\}.

3.2 Semantics of $\mathcal{AC}$

The semantics of $\mathcal{AC}$ is defined by using states and transitions.

A *state* σ is a pair of sets of fluent names $\langle \sigma^+, \sigma^- \rangle$ such that σ^+ and σ^- are disjoint, i.e., $\sigma^+ \cap \sigma^- = \emptyset$. Given a fluent name F and a state σ, we say that F holds in σ if $F \in \sigma^+$, F does not hold in σ if $F \in \sigma^-$, and it is not known whether F holds in σ otherwise. Given a fluent name F, we also say that $\neg F$ holds in σ if F does not hold in σ. Sometimes we also say that F is true in σ if $F \in \sigma^+$, F is false in σ if $F \in \sigma^-$, and F is undefined in σ otherwise.

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, into the set of states.

A *structure* is a pair (σ_0, Φ), where σ_0 is a state, called the *initial state* of the structure, and Φ is a transition function. For any structure $M = (\sigma_0, \Phi)$ 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 $\Phi(A_m, \Phi(A_{m-1}, \ldots, \Phi(A_1, \sigma_0) \ldots))$. A v-proposition of the form (1) is satisfied in a structure $M = (\sigma_0, \Phi)$ iff F holds in the state $\Phi(A_1; \ldots; A_m, \sigma_0)$.

We say that the execution of an action A in a state σ *immediately initiates* a fluent expression F if there is an e-proposition A **causes** F **if** $P_1, \ldots, P_m$ in the domain description such that for each $1 \leq i \leq m$, P_i holds in σ. Moreover, we say that the execution of A in σ *initiates* a fluent expression F if A immediately initiates F, or there is a $B \subseteq A$ such that execution of B in σ *immediately initiates* F and there is no C such that $B \subset C \subseteq A$, where execution of C in σ immediately initiates $\neg F$.

Let F be positive. When A initiates $\neg F$, we will say that A terminates F. We define two set-ranged functions as follows:

$$\begin{aligned} Initiate(A, \sigma) &= \{F \ : \ F \in \Sigma_f \text{ and } A \text{ initiates } F \text{ in } \sigma\} \\ Terminate(A, \sigma) &= \{F \ : \ F \in \Sigma_f \text{ and } A \text{ initiates } \neg F \text{ in } \sigma\} \end{aligned}$$

Note that $Initiate(A, \sigma)$ and $Terminate(A, \sigma)$ are not necessarily disjoint. We also need a set-ranged function $Cause(F, \sigma, A)$, (for fluent name F, state σ, and action expression A), which is defined to contain and only contain all the *set-inclusion minimal* subactions B of A satisfying: B immediately initiates F (or $\neg F$, resp.) in σ and there is no C such that $B \subset C \subseteq A$, where C immediately initiates $\neg F$ (or F, resp.) in σ. If $B \in Cause(F, \sigma, A)$, we will say that B is a cause for the change in truth values of the fluent name F in σ when A is done. The truth value of the fluent name F may change in two ways: from $true$ to $false$ and from $false$ to $true$. Note that the subaction B is set-inclusion minimal among all subactions which satisfy one of the two conditions above.

For later purpose we also need one more auxiliary functions:

$$\Delta(A, \sigma) \quad = \quad \bigcup_{B \in W} (Initiate(B, \sigma) \cup Terminate(B, \sigma))$$

where

$$W = \bigcup_{F \in Initiate(A,\sigma) \cap Terminate(A,\sigma)} Cause(F, \sigma, A)$$

Intuitively speaking, $\Delta(A, \sigma)$ denotes those fluents influenced by subactions of A which have conflicting effects. All of the fluents influenced by these subactions will be made undefined in the new situation resulting from doing A.

Definition 3.1 (Model) *A structure* (σ_0, Φ) *is a* model *of a domain description* D *iff*

- *Every v-proposition of* D *is satisfied in* (σ_0, Φ).
- *For every action expression* A *and every state* σ,

$$\Phi(A, \sigma) = \langle S^+, \ S^- \rangle$$

where

$$\begin{aligned} S^+ &= (\sigma^+ \cup Initiate(A, \sigma)) \setminus Terminate(A, \sigma) \setminus \Delta(A, \sigma) \\ S^- &= (\sigma^- \cup Terminate(A, \sigma)) \setminus Initiate(A, \sigma) \setminus \Delta(A, \sigma) \end{aligned}$$

A domain description is consistent *if it has a model. A domain description is* complete *if it has exactly one model. We will use* $Mod(D)$ *to denote the set of all models of* D.

Definition 3.2 (Entailment) *A v-proposition is* entailed *by a domain description* D *if it is satisfied in every model of* D.

All of D_{YSP}, D_{door}, and D_{soup} defined before are complete since they have only one model, respectively.

3.3 An alternative semantics

In this subsection we briefly introduce an alternative semantics, which is more skeptical than that in the last subsection. For the sake of simplicity, let F and P be two positive fluents. Consider

$$A \textbf{ causes } F \textbf{ if } P$$

F will be regarded as unchanged by A when A is performed in state σ, if $P \notin \sigma^+$. When $P \notin \sigma^+$, there are two possibilities: $P \in \sigma^-$ or $P \notin \sigma^-$. If $P \notin \sigma^+ \cup \sigma^-$, the truth value of P is not known. However, P may be actually true or false. If P were actually true, then F would be actually true after A is done in σ. The problem will arise when $F \in \sigma^-$. Based on this analysis, we have developed an alternative but more skeptical semantics which is a slight reduct from the semantics in the last subsection in the following way: When $F \in \sigma^-$ and $P \notin \sigma^+ \cup \sigma^-$, F will be regarded as *unknown* after A is done in σ instead of *unchanged* by disabling the inertia law on F in state σ. It seems to us that this alternative semantics is more skeptical and closer to reality. This alternative semantics is almost the same as the semantics in the previous section, and their difference is best illustrated by the following two examples. Let $D_1 = \{$ **initially** F. A **causes** F **if** $P\}$, and $D_2 = \{$ **initially** $\neg F$. A **causes** F **if** $P\}$. According to the new alternative semantics, F should remain to be true if A is done in the initial state in the domain D_1, while F should be unknown if A is done in the initial state in the domain D_2.

The detailed technical definition of this alternative semantics will be left out. In the following section, we will only use the semantics in the previous subsection.

4 FROM $\mathcal{AC}$ TO ABDUCTIVE LOGIC PROGRAMS

In order to automate temporal reasoning in $\mathcal{A}$, now we present a sound and complete translation from domain descriptions in $\mathcal{AC}$ to abductive logic programs. The familiarity with abductive logic programming is assumed. We write imm to abbreviate "immediately" in $imm\_initiates(A, F, S)$ and $imm\_terminates(A, F, S)$. In the translation, all the predicates are either self-explanatory or mirror the definitions of Subsection 3.2. The symbol *not* in logic programs denotes the negation-as-failure operator.

Let D be a domain description in $\mathcal{AC}$. The translation πD consists of a normal logic program and a set of constraints, defined as follows:

1. Auxiliary predicates about subactions:
   Assume that we are given standard rules for set-theoretical predicates $member(A, S)$, $subseteq(S_1, S_2)$. Sets can be represented by lists. Using these predicates we can define $subacteq(S_1, S_2)$ and $subact(S_1, S_2)$ about subactions in logic programming rules.
2. Initialization:
   $$is\_true(F, s_0) \quad \leftarrow \quad initially\_true(F). \tag{3}$$
   $$is\_false(F, s_0) \quad \leftarrow \quad initially\_false(F). \tag{4}$$
   where s_0 is a new symbol to denote the initial situation. The $initially\_true(F)$ and $initially\_false(F)$ are taken to be abducible predicates. If a fluent name F is abduced to be true (false, resp.) initially, then it is true (false, resp.) in the initial situation s_0.
3. Auxiliary Predicates:
   These auxiliary predicates will be used to define two main predicates $is\_true(F, S)$ and $is\_false(F, S)$, which indicate whether fluent F is true or false in situation S.

$initiates(A, F, S) \leftarrow imm\_initiates(A, F, S)$
$initiates(A, F, S) \leftarrow subacteq(B, A),$
$\quad imm\_initiates(B, F, S),$
$\quad not\ clip\_initiates(F, B, A, S).$
$terminates(A, F, S) \leftarrow imm\_terminates(A, F, S).$
$terminates(A, F, S) \leftarrow subacteq(B, A),$
$\quad imm\_terminates(B, F, S),$
$\quad not\ clip\_terminates(F, B, A, S).$
$causes(F, S, A, B) \leftarrow subacteq(B, A),$
$\quad imm\_initiates(B, F, S),$
$\quad not\ clip\_initiates(F, B, A, S),$
$\quad not\ clip\_cause1(F, B, A, S).$
$causes(F, S, A, B) \leftarrow subacteq(B, A),$
$\quad imm\_terminates(B, F, S),$
$\quad not\ clip\_terminates(F, B, A, S)$
$\quad not\ clip\_cause2(F, B, A, S).$
$delta(A, S, F) \leftarrow initiates(A, G, S),$
$\quad terminates(A, G, S), causes(G, S, A, B),$
$\quad initiates(B, F, S).$
$delta(A, S, F) \leftarrow initiates(A, G, S),$
$\quad terminates(A, G, S), causes(G, S, A, B),$
$\quad terminates(B, F, S).$
$clip\_initiates(F, B, A, S) \leftarrow subact(B, C),$
$\quad subacteq(C, A), imm\_terminates(C, F, S).$
$clip\_terminates(F, B, A, S) \leftarrow subact(B, C),$
$\quad subacteq(C, A), imm\_initiates(C, F, S).$
$clip\_cause1(F, B, A, S) \leftarrow subact(B, C),$
$\quad subacteq(C, A), imm\_initiates(C, F, S),$
$\quad not\ clip\_initiates(F, C, A, S).$
$clip\_cause2(F, B, A, S) \leftarrow subact(B, C),$
$\quad subacteq(C, A), imm\_terminates(C, F, S),$
$\quad not\ clip\_terminates(F, C, A, S).$

4. Main Predicates:
The following predicates are used to determine whether a fluent is true or false in a situation.

$is\_true(F, result(A, S)) \leftarrow is\_true(F, S),$
$\quad not\ terminates(A, F, S), not\ delta(A, S, F).$
$is\_true(F, result(A, S)) \leftarrow initiates(A, F, S),$
$\quad not\ terminates(A, F, S), not\ delta(A, S, F).$
$is\_false(F, result(A, S)) \leftarrow is\_false(F, S),$
$\quad not\ initiates(A, F, S), not\ delta(A, S, F).$
$is\_false(F, result(A, S)) \leftarrow terminates(A, F, S),$
$\quad not\ initiates(A, F, S), not\ delta(A, S, F).$

5. Domain-Specific Predicates
The syntax and semantics of the following predicates depend on domain descriptions. Let F be a fluent name, i.e., $F \in \Sigma_f$. Then, we write $Holds(F, S)$ to stand for $is\_true(F, S)$ and write $Holds(\neg F, S)$ for $is\_false(F, S)$.

- For each effect proposition a **causes** f **if** $p_1, \ldots, p_n$ in D, where f is positive, we have a logic programming rule:

$$imm\_initiates(a, f, S) \leftarrow Holds(p_1, S), \ldots, Holds(p_n, S).$$

- For each effect proposition a **causes** $\neg f$ **if** $p_1, \ldots, p_n$, where f is positive, we have a logic programming rule:

$$imm\_terminates(a, f, S) \leftarrow Holds(p_1, S), \ldots, Holds(p_n, S).$$

The integrity constraints, denoted by IC_D, are defined as follows: For each value proposition F **after** $A_1; \ldots; A_m$, we have:

$$Holds(F, result(A_1; \ldots; A_m, s_0)) \tag{5}$$

As we will use the predicate completion semantics, the above integrity constraint can be equivalently transformed into

$$false \leftarrow not\ Holds(F, result(A_1; \ldots; A_m, s_0)).$$

Note that we cannot abduce a fluent to be both true and false. For this purpose we add the following domain-independent constraint:

$$false \leftarrow is\_true(F, S_0), is\_false(F, S_0). \tag{6}$$

The literal $false$ is always interpreted as logical falsity, and all rules with $false$ as heads function as integrity constraints. The predicates $initially\_true(F)$ and $initially\_false(F)$ are taken as abducible predicates used to capture the incomplete knowledge about the initial situation. The semantics of πD is defined to be the union of the integrity constraints, the Clark Equality Theory, and the first-order theory obtained by completing all the non-abducible predicates [4]. The definition for abducible predicates $initially\_true(F)$ and $initially\_false(F)$ are left open. We will write $COMP(\pi D)$ to denote the semantics of of πD. The following two results justify the above semantics definition.

Proposition 4.1 *πD is an acyclic program with first-order constraints in the sense of [1].*

Corollary 4.2 *For the abductive logic program πD, $COMP(\pi D)$ coincides with its generalized stable model semantics [14] and generalized well-founded model semantics [21].*

The following two technical results are proved in the long version of this paper.

Theorem 4.3 (Soundness) *Let D be any domain description. For any value proposition Q, if $COMP(\pi D) \models \pi Q$, then D entails Q.*

Theorem 4.4 (Completeness) *Let D be a domain description. For any value proposition Q, if D entails Q, then $COMP(\pi D) \models \pi Q$.*

Since our translation is both sound and complete, we can reduce temporal reasoning in $\mathcal{AC}$ to abductive query evaluaiton in logic program πD. The work reported in this paper has been experimented with the latest version of the REVISE system [5]. The use of the REVISE system is not essential, and other abductive query evaluaiton procedures for normal logic programs such as SLDNFA [6] and that in [23] should also work. If for every fluent name F, one and only one of **initially** F and **initially** $\neg F$ appears in a domain description, then PROLOG can be used for temporal projection after (3), (4), and (6) are removed and the integrity constraitns are represented only in the form of (5). For example, temporal reasoning in the Yale Shooting domain D_{YSP} can be done in PROLOG. Furthermore, if we introduce more propositions, say observation propositions, we can abductively diagnose action domains when there is a discrepancy between predicted propositions and observed propositions. For the sake of space limitation, we will not go into deeper discussions.

5 RELATED WORK

There have been many reports on temporal reasoning. It is extremely difficult, if not impossible, to compare our work with all of them.

With respect to the action language $\mathcal{A}$ and its extensions, there are four major approaches to the translation of $\mathcal{A}$-like action languages into logic programs . In what follows we will briefly compare our work with [11, 7, 8, 3].

In [11] Gelfond and Lifschitz proposed the action language $\mathcal{A}$, presented a translation π_{GL} from $\mathcal{A}$ to extended logic programs with answer sets as semantics, and showed that their translation π_{GL} is sound for domain descriptions without similar effect propositions, but not complete. The main differences between [11] and ours lie in the following facts: (1) Concurrent actions are not directly supported in $\mathcal{A}$; (2) Fluents are three-valued in $\mathcal{AC}$, while they are two-valued in $\mathcal{A}$. As a consequence, there may not be any state transition satisfying e-propositions of some domain descriptions of $\mathcal{A}$; (3) π_{GL} and π use different logic programming paradigms and semantics: π_{GL} uses extended logic programs and answer set semantics, while π uses abductive normal logic programs and predicate completion semantics. Since πD is acyclic, its predicate completion semantics coincides with generalized stable model semantics and well-founded model semantics; (4) Since our translation is both sound and complete for all domain descriptions, we can immediately combine our work with Kartha's work [15] and relate the abductive normal logic program πD to three other formalisms of reasoning about actions, proposed by Pednault [20], Reiter [22] and Baker [2].

In [7] Denecker and Schreye elaborated on Gelfond and Lifschitz' original work, and proposed a translation π_{DS} from domain descriptions in $\mathcal{A}$ to abductive normal logic programs. The main differences between [7] and our work are: (1) Concurrent actions are not directly supported in $\mathcal{A}$; (2) Fluents are three-valued in $\mathcal{AC}$, while they are two-valued in $\mathcal{A}$. (3) Our translation is complete for any domain description, but π_{DS} is complete only for e-consistent domain descriptions. We should point out that our logic programming paradigm is the same as that of [7]. Another approach, similar to [7], was proposed by Dung [8], who proposed a translation from $\mathcal{A}$ to logic programs with a special semantics similar to the completion semantics. Dung treats the fluent symbol and its negation symmetrically. In particular, Dung considered its application in database integrity constraints in the presence of updates. Concurrent actions are not considered in Dung [8].

In [3] Baral and Gelfond made a first step to extend $\mathcal{A}$ with concurrent actions. The semantical difference between $\mathcal{AC}$ and $\mathcal{A}_{BG}$ has been discussed in Section 2. Another difference lies in the translation of domain descriptions into logic programs: π_{BG} inherits the incompleteness of π_{GL}, but our translation is complete for any domain description.

6 CONCLUSIONS

We have analyzed two proposals for semantics of concurrent actions [19, 3], and given a new semantics to common-sense concurrent actions. We have also extended $\mathcal{A}$ with concurrent actions, and presented a translation from domain descriptions in the resulting new action description language to abductive logic programs, given the soundness and completeness theorems of the translation. We have also compared ours with related work. Our method is applicable to the temporal projection problem with incomplete information, as well as to reasoning about the past. Many benchmark examples in the framework of this paper have been experimented with the latest version [5] of the REVISE system. The future work will focus on extensions of $\mathcal{AC}$ (by allowing, e.g, constraints, durative actions, etc.) and its use in practice.

ACKNOWLEDGEMENTS

This work was supported by a post-doctoral fellowship from JNICT under PRAXIS XXI/BPD/4165/94 to the first author, and JNICT PROLOPPE project under PRAXIS/3/31/TIT/24/94.

REFERENCES

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

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

[3] Baral, C. and Gelfond, M., Representing concurrent actions in extended logic programming, *Proc. of IJCAI'93*, 1993, pp. 866–871

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

[5] Damásio, C. V., Pereira, L.M., and Wolfgang, N. , REVISE: An Extended Logic Programming System for Revising Knowledge Bases, *Proc. of KR'94*, 1994

[6] Denecker, M., De Schreye, D., SLDNFA: an abductive procedure for normal abductive programs, *Logic Programming: Proc. of 1992 Int'l Joint Conference and Symposium*, . Apt, ed., MIT Press, 1992, 686-700

[7] Denecker, M. and de Schreye, D., Representing incomplete knowledge in abductive logic programming, *Logic Programming: Proc. of the 1993 Int'l Symposium*, MIT Press, 1993, 147–163

[8] Dung, P.M., Representing actions in logic programming and its application in database updates, *Proc. of ICLP 93*, MIT Press, 1993, 222–238

[9] Gelfond, M., and Lifschitz, V., The stable model semantics for logic programming, In. *Proc. of ILPS'88*, MIT Press, 1988, 1070-1080

[10] Gelfond, M. and V. Lifschitz, Logic programs with classical negation, *Proc. of ICLP'90*, MIT Press, 1990, pp. 579-597

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

[12] Gelfond, M., Lifschitz, V., and Rabinov, A., What are the limitations of the situation calculus?, Automated Reasoning: Essays in Honor of Woody Bledsoe, R. Moore (ed.), 1991, 167-179

[13] Hanks, S, and McDermott, D., Nonmonotonic logics and temporal projection, *Artificial Intelligence*, 35, 1988, 165–195

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

[15] Kartha, G.N., Soundness and completeness theorems for three formalizations of action, *Proc. IJCAI93*, 1993, 712–718

[16] Kartha, G.N. and Lifschitz, V., Actions with indirect effects: preliminary report, *Proc. of KR94*, 1994, 341–350

[17] Lifschitz, V., Nested Abnormal Theories, Uni. of Texas at Austin, manuscript, 1994

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

[19] F. Lin and Y. Shoham, Concurrent actions in the situation calculus, *Proc. of AAAI-92*, 1992, pp.590-595

[20] Pednault, E. P. D., ADL: Exploring the middle ground between STRIPS and the situation calculus, *Proc. of KR'89*, R. J. Brachman, H. Levesque, R. Reiter (eds.), Morgan Kaufmann Publishers, Inc., 324–332

[21] Pereira, L. M., Alferes, J. J., and Aparício, J. N., Nonmonotonic Reasoning with Well Founded Semantics, In: K. Furukawa (ed.), *Proc. of ICLP'91* , MIT Press, 1991, 475-489

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

[23] Satoh, K., Iwayama, N., A query evaluation method for abductive logic programming *Logic Programming: Proc. of 1992 Int'l Joint Conference and Symposium*, 1992, 671-685