Representing and Reasoning about Concurrent Actions with Abductive Logic Programs*

Renwei Li and Luís Moniz Pereira
Centre for Artificial Intelligence (CENTRIA)
Universidade Nova de Lisboa
2825 Monte da Caparica, Portugal
{renwei | lmp}@fct.unl.pt

Contents

*Accepted by *Annals of Mathematics and Artificial Intelligence*, Special Issue in Honor of Michael Gelfond, 1997.

Abstract In this paper we extend Gelfond and Lifschitz' action description language $\mathcal{A}$ with concurrent actions and observation propositions to describe the predicted behaviour of domains of (concurrent) actions and actually observed behaviour, respectively, without requiring that the actually observed behaviour of a domain of actions be consistent with its predicted behaviour. We present a translation from domain descriptions and observations in the new action language to abductive normal logic programs. The translation is shown to be both sound and complete. From the stand-point of model-based diagnosis, in particular, we discuss the temporal explanation of inferring actions from fluent changes at two different levels, namely, at the domain description level and at the abductive logic programming level. The method is applicable to the temporal projection problem with incomplete information, as well as to the temporal explanation of inferring actions from fluent changes.

1 Introduction

There have been many efforts in formalisms for reasoning about actions and changes. Among others, the situation calculus [47] and the event calculus [35] are two examples. The situation calculus is a very general framework for reasoning about actions and changes. The event calculus as a theory for reasoning about events in a logic programming framework is based in part on the situation calculus, but seeks to combine the deductive power of logic with the computational power of logic programming [35]. Recent investigations have shown that the situation calculus and event calculus are closely related to each other [36, 9].

The main tasks involved in reasoning about actions and changes are as follows: Given a domain description with fluents holding and actions happening at certain times or states, we want to reason forward and backward in time, inferring actions from changes of fluents and inferring changes of fluents from actions. One of main difficulties here lies in intuitively correct and computationally efficient solutions to the frame problem: how to deal with fluents which are not changed by a particular action [47].

The earliest nonmonotonic solution to the frame problem by using naive circumscription or normal defaults was jeopardized by the so-called Yale Shooting Problem (YSP) identified by Hanks and McDermott [28]. Since then various approaches have been proposed to solve the YSP. We list only a few among others:

- Chronological minimization. It was proposed independently by Kautz [34], Lifschitz [43], and Shoham [66]. The general idea is to prefer to delay changes as long as possible when predicting the future. Kautz [34] showed that chronological minimization cannot handle the explanation-type problems such as the Stolen Car Problem. Modifications and improvements of chronological minimization have been proposed in literature. For example, Sandewall [62] proposed a two-staged method, called *filter preferential entailment*, which first selects the models that maximize persistence expectations, and then filters out those models that disagree with the observations.

- Causal minimization. It was proposed independently by Haugh [29] and Lifschitz [40]. The general idea is to make causality explicit and assume that every fluent change has to be caused by some action. Causal minimization prefers sequences of world states in which actions only have explicitly stated effects. Causal minimization can give desired results for temporal prediction, but it only works in frameworks where all the events are known and does not allow us to write domain-specific axioms in unrestricted situation calculus.

- Baker's circumscriptive theory [5, 6]. Minimization of change by using circumscription was initially proposed by McCarthy [48], where the abnormality ab is minimized with $holds$ varied. As pointed out by Hanks and McDermott, the Yale Shooting Problem will arise since there will be two minimal models. Instead of McCarthy's naive circumscription policy, Baker circumscribes ab with $result$ and S_0 varied. Baker showed that his approach is more robust than other approaches such as causal minimization and chronological minimization in the sense that it can handle variations of the original YSP. Specifically, it is distinguished by its capability to deal with reasoning backward in time and with indirect effects of actions (ramifications). Closely related work can be found in, e.g., [6, 12, 32, 39]. In this approach Baker needs the axioms for the existence of situations (there is a situation for each of all the possible combinations of fluents), and domain closure axioms. Recently Kartha [31] has given two interesting counter-examples to Baker's minimization approach, when applied to nondeterministic actions, and showed how to rectify it.

- The use of semi-normal defaults. Instead of using normal defaults to solve the frame problem, Morris [49] proposed to use semi-normal defaults, and thus effectively avoided the anomalous extension problem.
- The use of autoepistemic logic. This was proposed by Gelfond [21]. Both the use of semi-normal defaults and the use of autoepistemic logic have substantially influenced the logic programming approach to formalizing actions and changes.
- Logic programming. Instead of using various circumscription policies, a logic program uses an operator called negation-as-failure (*not*). The YSP can be expressed as a logic program (see, for example, [20, 4]). The resulting logic program can be proved to be locally stratified and thus has a unique perfect model [57] as intended[1].

Because there are so many different approaches to reasoning about actions and changes, it is very difficult, if not impossible, to compare and contrast them with each other, although they share a few basic assumptions.

Instead of constructing new special-purpose formalisms, Gelfond and Lifschitz [24] proposed a simple high-level declarative action description language $\mathcal{A}$ for reasoning about actions and intended to use it as a basis for systematic development of theories of actions and their effects[2]. In [24] Gelfond and Lifschitz presented a translation from $\mathcal{A}$ to extended logic programs with answer sets as semantics [22, 23], and proved its soundness. Kartha proposed three translations of $\mathcal{A}$ into three formalisms of reasoning about actions proposed by Pednault [50], Reiter [61] and Baker [5], respectively. Pednault's and Reiter's formalisms are actually first-order monotonic formalisms, while Baker's method is based on circumscription. Kartha also showed that the soundness and completeness theorems hold for all his three translations. Kartha and Lifschitz [33] also considered how to deal with ramifications in $\mathcal{A}$ by using nested abnormality theory [41]. Baral and Gelfond [7, 8] extended the action language $\mathcal{A}$ with concurrent actions and presented two translations into extended logic programs and disjunctive logic programs, respectively. Dung presented a translation of domain descriptions in $\mathcal{A}$ into abductive logic programs with database updates as applications [18]. Denecker and de Schreye proposed another much simpler translation of $\mathcal{A}$ into abductive normal logic programs with constraints [17] and showed that their translation is both sound and complete with respect to the predicate completion semantics [11] of abductive normal logic programs. All these results impress on us that the action description language $\mathcal{A}$ is really 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 examine the action description language $\mathcal{A}$, and analyze why there are no models for some domains such as the Stolen Car Problem (SCP) [34]. We will give a modification of $\mathcal{A}$ by adding concurrent actions and observation propositions so as to describe actual observed behaviour of domains of actions. By using domain description propositions and domain observation propositions we can make a distinction between the predicted behaviour and actually observed behaviour of domains of (concurrent) actions, without requiring that they be consistent with each other. We will present a translation from domain descriptions and observations to abductive logic programs,

[1] The program is even acyclic [4]. Notice that every acyclic program can be shown to be also locally stratified.

[2] Another action description language was proposed by Pednault [50, 51]. Sandewall [63, 64] presented another systematic approach to representing and reasoning about actions and changes in his *fluents-features framework* (FFF).

prove the soundness and completeness, and compare our work with others. In particular, from the stand-point of model-based diagnosis we will discuss the temporal explanation of inferring actions from fluent changes at two different levels, namely, at the domain description level and at the abductive logic programming level, respectively, and relate them to each other. Our method is applicable to the temporal projection problem with incomplete information, as well as to the temporal explanation of inferring actions from fluent changes. A short version of the first part of this paper on semantics of our action description language and its translation into logic programs has been published in [37].

The rest of this paper is organized as follows. In Section 2 we discuss the Stolen Car Problem, analyze why there are no models for it in $\mathcal{A}$, and motivate the development of the rest of this paper. In Section 3 we give a modified $\mathcal{A}$ and its semantics. In Section 4 we briefly introduce abductive logic programming with constraints. In Section 5 we present a translation from domain descriptions in the new action language to abductive logic programs. In Section 6 we show our translation is sound and complete. In Section 7 we discuss the temporal explanation from the stand-point of model-based diagnosis. In Section 8 we compare our work with other work. Finally, in Section 9, we conclude this paper with a few remarks.

2 A motivating example

In this section we assume that the reader is familiar with the action description language $\mathcal{A}$. The rest of the paper is self-contained. Without difficulty with understanding the remainder of the paper the reader can skip over this section.

Now we consider the Stolen Car Problem (SCP) motivated by an example from Kautz [34]. The SCP domain has one fluent name *Stolen* and two action names *Wait* and *Steal*[3]. Gelfond and Lifschitz [24] use the following two propositions to describe the SCP domain D_1:

$$\textbf{initially}\ \neg Stolen$$
$$Stolen\ \textbf{after}\ \ Wait; Wait; Wait$$

We add a new effect proposition for the action *Steal*:

$$Steal\ \textbf{causes}\ \ Stolen$$

Whether the new effect proposition is added or not, it can be shown that the SCP domain description D_1 is inconsistent, and thus has no model.

On the other hand, it seems that the above three propositions roughly correspond to a stolen-car story: In the morning someone parked the car in a parking lot (**initially** $\neg Stolen$). Then he went to work in the morning (roughly represented by *Wait*), had lunch at noon (roughly represented by *Wait*), worked again in the afternoon (roughly represented by *Wait*). But in the evening he found that the car is missing (roughly represented by $Stolen$ **after** $Wait; Wait; Wait$). He certainly knows that if a thief steals his car then his car will be stolen ($Steal$ **causes** $Stolen$). There is nothing wrong with the above story; we can find some true cases in our reality, which suggests that we should have a model for the SCP domain.

Our stance is that what is wrong with the domain description D_1 is that we only have incomplete knowledge about what has happened in the past. Our intuition is that the action *Steal* has happened *at the same time* as one of the three *Wait* actions. Thus, one of the situations $Result(Wait, S0)$, $Result(Wait; Wait, S0)$, $Result(Wait; Wait; Wait, S0)$ does not correctly represent the real situation. By the intuition of [47] the situation $Result(a, s)$ should denote a snapshot of the universe at an instant in a period of time. In the stolen-car story, one of the situations $Result(Wait, S0)$, $Result(Wait; Wait, S0)$, $Result(Wait; Wait; Wait, S0)$ never becomes

[3] The action name *Steal* was not introduced in [24], since it is not necessary there. The reader had better regard the action *Wait* as an action which has effects on fluents which are not of interest. For example, the first *Wait* in $Stolen$ **after** $Wait; Wait; Wait$ may denote "to work in the morning", the second may denote "to have lunch" and the third may denote "to work in the afternoon".

true, since what becomes true is some other situation, say $Result(Wait||Steal, S)$, where we use $Wait||Steal$ to mean that actions $Wait$ and $Steal$ happen at the same time.

Our position here is different from those in existing approaches to temporal explanation championed by, e.g. [5, 6, 12, 42, 69]; they would conclude that some of the three $Wait$ actions are abnormal. It is the abnormality of some of the three $Wait$ actions that is consistent with the car being missing. But there are no satisfactory explanations for why one of the three $Wait$ actions is abnormal. On the other hand, Shanahan [68] showed that there is a $Steal$ action between the morning when one parked the car and the evening when he left his office. But the $Wait$ actions are ignored. We think that what is the case in reality is that $Steal$ and $Wait$ have happened simultaneously at some time in the day, though $Steal$ was not observed by the owner of the car.

Now let's see whether our intuition works. Substituting $Wait||Steal$ for one of the three actions $Wait$ in the value proposition $Stolen$ **after** $Wait; Wait; Wait$, say the first one, we then have something like this:

$$Stolen \textbf{ after } Wait||Steal; Wait; Wait$$

Using a suitable semantics for $\mathcal{A}$ extended with concurrent actions, we then have a model for the SCP domain.

The above analysis leads to the following two basic problems:

- We need some means to extend the action language $\mathcal{A}$ with concurrent actions.
- We need some means to express incomplete knowledge about the past history.

The extension of $\mathcal{A}$ with concurrent actions has its own rightful development in domain descriptions as argued by Baral and Gelfond [7]. In this paper we will simply extend $\mathcal{A}$ with concurrent actions, though in a way somewhat different from that of [7] both in the semantics and the translation into logic programs. But our main innovation in this paper concerns the way we address and solve the second problem above, given an adequate solution to the first.

For representation of incomplete knowledge about the past history we need to extend the action language $\mathcal{A}$, since its value propositions of the form F **after** $A_1, A_2, \ldots, A_m$ are only appropriate for complete knowledge about what will be true if actions $A_1, A_2, \ldots, A_m$ exactly (no more, no less) are done in sequence. We will introduce a new kind of propositions, called *observation propositions*, whose syntax is as follows:

$$\textbf{observed } F \textbf{ after } A_1; A_2; \ldots; A_m$$

The observation proposition of the above form simply means that it is observed that F is true after we have known actions $A_1; A_2; \ldots; A_m$ have been done. Because of the incompleteness of observations, some other yet unknown actions may have also occurred while actions $A_1, A_2, \ldots, A_m$ were done. To resolve the clash between predictions and observations, we will employ abduction to diagnose the occurrences of unobserved concurrent actions. This allows actually explaining the observations, not just making the predictions and observations consistent by assuming abnormalities of actions.

The technical details will be found in subsequent sections.

3 An action description language and its semantics

We will extend Gelfond and Lifschitz' action language $\mathcal{A}$ with concurrent actions and observations. The resulting language is denoted by $\mathcal{A}_{\mathcal{CO}}$, where the subscript C is for concurrent actions and O for observations. Given a domain, we will make a distinction between its description and its observation. A domain description states how the domain should behave, while an observation of it tells us some incomplete past history. Usually, domain observations are employed when we need to infer what actions have happened from an incomplete past history. In this section we will only consider, for now, the domain description part of $\mathcal{A}_{\mathcal{CO}}$, which is just an enrichment of $\mathcal{A}$ with concurrent actions.

3.1 Domain description syntax of $\mathcal{A}_{\mathcal{CO}}$

We begin with the alphabet Σ of a domain. The alphabet Σ of a domain is defined to be two disjoint non-empty sets of symbols Σ_f and Σ_a, called *fluent names* and *atomic action names*, respectively.

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[4]. 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 the action expression $\{A\}$ if A is an atomic action name. We intend an action expression to denote 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 $\mathcal{A}_{\mathcal{CO}}$ there are three kinds of propositions: value propositions, effect propositions, and observation propositions, simply called v-propositions, e-propositions, o-propositions. The observation propositions will be introduced in Section 7.

A v-proposition is defined to be a statement of the form

$$F \textbf{ after } A_1; \ldots; A_m \tag{1}$$

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

$$\textbf{initially } F$$

An e-proposition is defined to be a statement of the form

$$A \textbf{ causes } F \textbf{ if } P_1, \ldots, P_n \tag{2}$$

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

$$A \textbf{ causes } F$$

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

Example 3.1 (Stolen Car Problem [34]) *Let the alphabet of the Stolen Car Problem domain includes Steal and Wait as atomic actions, and Stolen as a fluent. Its domain description D_{scp} includes the following propositions:*

$$\textbf{initially } \neg Stolen$$
$$Steal \textbf{ causes } Stolen$$

Example 3.2 (Door) *Consider a scenario in which there is a door and a light. The fluent names are Closed and On, and the action names are Open, Close, and Switch. By doing action Open one can make sure the door is not closed, and by doing action Close one can make sure the door is closed. The action Switch just changes the status of the light. The domain description D_{door} includes the following propositions:*

$$\textbf{initially } \neg Closed$$
$$\textbf{initially } On$$

[4] One may argue that it is sometimes impossible for two atomic actions to happen at the same time in practice. In this case, the definition of concurrent actions can be defined in a non-monotonic way: normally a non-empty finite set of atomic action names is a concurrent action unless stated otherwise. In the following discussion, what matters is the concept of subactions, sometimes denoted by the set inclusion notation $\subseteq$. Suitably re-defining the concept of subactions, the results of this paper can be easily adapted.

$$\begin{array}{l} Open\ \mathbf{causes}\ \ \neg Closed \\ Close\ \mathbf{causes}\ \ Closed \\ Switch\ \mathbf{causes}\ \ On\ \mathbf{if}\ \ \neg On \\ Switch\ \mathbf{causes}\ \ \neg On\ \mathbf{if}\ \ On \end{array}$$

Example 3.3 (Mary's Soup [25]) *Mary wants to lift a bowl of soup to serve a guest. Whenever she lifts the bowl with only one hand, she will spill the soup to the floor. But when she lifts the bowl with two hands, she will not spill the soup. Suppose the soup is not spilled initially. Let the fluent names be Spilled, and action names* $Lift\_Left$ *and* $Lift\_Right$*. Then the domain description* D_{soup} *is described as follows:*

$$\begin{array}{l} \mathbf{initially}\ \ \neg Spilled \\ Lift\_Left\ \mathbf{causes}\ \ Spilled \\ Lift\_Right\ \mathbf{causes}\ \ Spilled \\ \{Lift\_Left,\ Lift\_Right\}\ \mathbf{causes}\ \ \neg Spilled \end{array}$$

When $\mathcal{A}_{\mathcal{CO}}$ is used to describe a domain, the following informally stated postulates are also implicitly used:

- Changes in the values of fluents can only be caused by execution of action expressions defined from atomic action names.
- Effects of an atomic action are only those specified by e-propositions.
- Effects of a concurrent action are the union of those specified by e-propositions and those inherited from its subactions, but only if not contrary to any e-proposition of their superactions which are in turn subactions of the concurrent action. However, if there are two subactions, neither of which contains the other and which have conflicting effects, then for these two subactions the fourth postulate is applied.
- If there is a fluent which is both initiated and terminated by two subactions of an action, neither of which contains the other, then all fluents effectively initiated and terminated by these subactions will be regarded as undefined in the new situation resulting from doing the action.

The above four postulates will be made clear and precise in the definition of the semantics of domain descriptions in $\mathcal{A}_{\mathcal{CO}}$.

The first postulate above is the same as that made in [24, 7, 8], among others. It is actually related to the so-called common-sense inertia law [24]. Informally, it says that a fluent keeps unchanged after occurrence of an action if it is not changed by it. The second postulate above is also same as that assumed in [24, 7]. The third postulate, called inheritance postulate, is about the effects of concurrent actions. By the third postulate a concurrent action not only have effects explicitly specified by its e-propositions, but also can inherit effects from its sub-actions. But if there are two subactions, neither of which contains the other and which have conflicting effects, the effects of the two subactions cannot be inherited by the concurrent action. The effect relationship between an action and its subaction was first observed by Gelfond, Lifschitz and Robinov in their pioneering work on the expressiveness of the situation calculus [25], and explicitly made clear and precise by Lin and Shoham [45] and Baral and Gelfond [7]. We will return to it later. The fourth postulate is new. Intuitively, we say an action A *initiates* (*terminates*) a fluent F if A makes F true (false) when A is done. The concepts of *initiation* and *termination* will be formally defined by two set-ranged functions $Initiate(A, \sigma)$ and $Terminate(A, \sigma)$ in the next subsection. In our framework we will regard fluents as three-valued. This postulate will be used to resolve conflicting effects of an action. For example, let a domain description D_2 include

$$\begin{array}{l} a\ \mathbf{causes}\ \ f \\ a\ \mathbf{causes}\ \ \neg f \\ a\ \mathbf{causes}\ \ g \end{array}$$

Note that a is also a subaction of itself. By the fourth postulate, fluents f and g will be regarded as undefined in the resulting situation after a is done. Note that we have taken a conservative approach: not only is f undefined but also is g. This postulate is particularly useful for evaluation of effects of concurrent actions when its subactions have conflicting effects. For example, when someone opens a door while another closes it at the same time, will the door be open or closed? In our framework, we will regard the status of the door as undefined. As another example, consider the following domain description D_3:

$$\begin{array}{ll} a \textbf{ causes} & f \\ a \textbf{ causes} & g \\ b \textbf{ causes} & \neg f \\ c \textbf{ causes} & \neg f \\ c \textbf{ causes} & \neg h \\ d \textbf{ causes} & i \end{array}$$

Now let's consider the concurrent action $\{a, b, c, d\}$. Since a and b are two subactions of $\{a, b, c, d\}$, neither of which contains the other and which have conflicting effects (a tries to initiate f and b tries to initiate $\neg f$), their effects cannot be inherited by $\{a, b, c, d\}$ according to the third postulate, instead f and g are regarded as undefined in the resulting situation by the fourth postulate. Analogously, for two subactions a and c of $\{a, b, c, d\}$, the effects of a and c cannot be inherited by $\{a, b, c, d\}$, instead f, g and h are regarded undefined in the resulting situation. Thus, after $\{a, b, c, d\}$ is done, i is true, and f, g, h are undefined.

The semantics of domain descriptions will be discussed in the next subsection.

3.2 Semantics of domain descriptions in $\mathcal{A}_{\mathcal{CO}}$

The semantics of $\mathcal{A}_{\mathcal{CO}}$ 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 $\Phi(A_1; \ldots; A_m, \sigma_0)$. In particular, the v-proposition **initially** F is satisfied in M iff F holds in the initial state σ_0. Note that a fluent expression may be positive or negative. By saying $\neg G$ (with G being a fluent name) holds in $\Phi(A_1; \ldots; A_m, \sigma_0)$ we actually mean G does not hold in $\Phi(A_1; \ldots; A_m, \sigma_0)$.

We say that the execution of an action A in a state σ *immediately initiates*[5] 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 $1 \le i \le m$, P_i holds in σ. Given an action expression A and a state σ, we can always say whether or not a particular fluent expression is immediately initiated by A in σ. Moreover, we say that the execution of an action A in a state σ *initiates* a fluent expression F if

- A immediately initiates F, or

[5] In [7] the term *causes* is used instead of *initiates*. We reserve *causes* for another purpose in this paper.

- 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 $\neg F$, where by $\neg\neg G$ we mean G.

Let F be positive. By saying that A initiates F we mean that A makes F true, and by saying that A initiates $\neg F$ we mean that A makes F false. From now on, we will make a distinction between them: when A initiates $\neg F$, we will say that A terminates F. We define two set-ranged functions as follows:

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

Given an action expression A and a state σ, since it is decidable whether or not a particular fluent expression is immediately initiated by A in σ, it is also decidable whether or not a particular fluent expression is initiated by A in σ. Thus, the above definition is well-defined.

Note that $Initiate(A, \sigma)$ and $Terminate(A, \sigma)$ are not necessarily disjoint. Sometimes two atomic actions can immediately initiate two complementary fluents. For example, let $Open$ and $Close$ be two actions to denote opening and closing a door, respectively. Let $Closed$ denote the status of the door. Stimulate

$$\begin{aligned} &Open \ \textbf{causes} \ \neg Closed \\ &Close \ \textbf{causes} \ Closed \end{aligned}$$

Then we would have

$$\begin{aligned} Initiate(\{Open, Close\}, \langle \emptyset, \{Closed\} \rangle) &= \{Closed\} \\ Terminate(\{Open, Close\}, \langle \emptyset, \{Closed\} \rangle) &= \{Closed\} \end{aligned}$$

For the later purpose we also need another 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 in σ and there is no C such that $B \subset C \subseteq A$, where C immediately initiates $\neg F$ in σ; or
- B immediately initiates $\neg F$ in σ and there is no C such that $B \subset C \subseteq A$, where C immediately initiates F 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 example, let a domain description include

$$\begin{aligned} &a \ \textbf{causes} \ f \\ &\{a, b\} \ \textbf{causes} \ \neg f \\ &\{a, b, c\} \ \textbf{causes} \ f \\ &\{a, b, c, d\} \ \textbf{causes} \ f \\ &e \ \textbf{causes} \ \neg f \end{aligned}$$

Let σ be any state. Then, by the above definition we have

$$Cause(f, \sigma, \{a, b, c, d, e\}) = \{\{a, b, c\}, e\}$$

In this example, although $\{a, b, c, d\}$ satisfies one of the two conditions above, but it is not set-inclusion minimal since its proper subset $\{a, b, c\}$ also satisfies one of the two conditions above. Moreover, neither a nor $\{a, b\}$ satisfies one of the two conditions above.

For later purpose we also need one more auxiliary functions:

$$\Delta(A,\sigma) \;=\; \bigcup_{F \in Initiate(A,\sigma) \cap Terminate(A,\sigma),\, B \in Cause(F,\sigma,A)} (Initiate(B,\sigma) \cup Terminate(B,\sigma))$$

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. The set-ranged functions defined above will be illustrated by the following two examples.

Example 3.4 *Consider the domain description D_{door} of Ex. 3.2. In the rest of this example we will write O for Open, C for Close, and S for Switch. Let $\sigma = \langle\{On\},\{Closed\}\rangle$. Then, we have:*

$$\begin{array}{rclrcl}
Initiate(O,\sigma) & = & \emptyset & Terminate(O,\sigma) & = & \{Closed\} \\
Initiate(C,\sigma) & = & \{Closed\} & Terminate(C,\sigma) & = & \emptyset \\
Initiate(S,\sigma) & = & \emptyset & Terminate(S,\sigma) & = & \{On\} \\
Initiate(\{O,C\},\sigma) & = & \{Closed\} & Terminate(\{O,C\},\sigma) & = & \{Closed\} \\
Initiate(\{O,S\},\sigma) & = & \emptyset & Terminate(\{O,S\},\sigma) & = & \{Closed, On\} \\
Initiate(\{C,S\},\sigma) & = & \{Closed\} & Terminate(\{C,S\},\sigma) & = & \{On\} \\
Initiate(\{O,C,S\},\sigma) & = & \{Closed\} & Terminate(\{C,C,S\},\sigma) & = & \{Closed, On\} \\
Cause(Closed,\sigma,O) & = & \{O\} & Cause(Closed,\sigma,C) & = & \{C\} \\
Cause(Closed,\sigma,S) & = & \emptyset & Cause(Closed,\sigma,\{O,C\}) & = & \{O,C\} \\
Cause(Closed,\sigma,\{O,S\}) & = & \{O\} & Cause(Closed,\sigma,\{C,S\}) & = & \{C\} \\
Cause(Closed,\sigma,\{O,C,S\}) & = & \{O,C\} & Cause(On,\sigma,O) & = & \emptyset \\
Cause(On,\sigma,C) & = & \emptyset & Cause(On,\sigma,S) & = & \{S\} \\
Cause(On,\sigma,\{O,C\}) & = & \emptyset & Cause(On,\sigma,\{O,S\}) & = & \{S\} \\
Cause(On,\sigma,\{C,S\}) & = & \{S\} & Cause(On,\sigma,\{O,C,S\}) & = & \{S\} \\
\Delta(O,\sigma) & = & \emptyset & \Delta(C,\sigma) & = & \emptyset \\
\Delta(S,\sigma) & = & \emptyset & \Delta(\{O,S\},\sigma) & = & \emptyset \\
\Delta(\{C,S\},\sigma) & = & \emptyset & \Delta(\{O,C\},\sigma) & = & \{Closed\} \\
\Delta(\{O,C,S\},\sigma) & = & \{Closed\} & & &
\end{array}$$

Note that the fluent Closed is both initiated and terminated by $\{O,C\}$ and $\{O, C, S\}$, respectively. The fluent Closed will be left undefined in the resulting situation after $\{O,C\}$ and $\{O, C, S\}$ are done, respectively.

Example 3.5 *Consider the domain description D_{soup} of Ex. 3.3. In the rest of this example we will write LL for Lift_Left and LR for Lift_Right. Let $\sigma = \langle\emptyset,\{Spilled\}\rangle$. Then, we have:*

$$\begin{array}{rclrcl}
Initiate(LL,\sigma) & = & \{Spilled\} & Terminate(LL,\sigma) & = & \emptyset \\
Initiate(LR,\sigma) & = & \{Spilled\} & Terminate(LR,\sigma) & = & \emptyset \\
Initiate(\{LL,LR\},\sigma) & = & \emptyset & Terminate(\{LL,LR\},\sigma) & = & \{Spilled\} \\
Cause(Spilled,\sigma,LL) & = & \{LL\} & Cause(Spilled,\sigma,LR) & = & \{LR\} \\
Cause(Spilled,\sigma,\{LL,LR\}) & = & \{\{LL,LR\}\} & \Delta(LL,\sigma) & = & \emptyset \\
\Delta(LR,\sigma) & = & \emptyset & \Delta(\{LL,LR\},\sigma) & = & \emptyset
\end{array}$$

Although both Lift_Left (LL) and Lift_Right (LR) can initiate Spilled if they are done independently, but $\{Lift\_Left, Lift\_Right\}$ does not initiate Spilled.

Now we are ready to define models of domain descriptions.

Definition 3.6 (Models of of Domain Descriptions) *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,*

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

where

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

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.

It can be shown that different models of the same domain description can differ only by their initial states, as in [24] .

Definition 3.7 (Entailment) *A v-proposition is entailed by a domain description D if it is satisfied in every model of D.*

Example 3.8 *Consider the domain description D_{scp} of Ex. 3.1. D_{scp} is complete since it has only one model:*

$$\begin{aligned}
\sigma_0 &= \langle \emptyset, \{Stolen\}\rangle \\
\Phi(Wait,\sigma) &= \sigma \\
\Phi(Steal,\sigma) &= \langle \{Stolen\}, \emptyset\rangle \\
\Phi(\{Steal, Wait\},\sigma) &= \langle \{Stolen\}, \emptyset\rangle
\end{aligned}$$

Example 3.9 *Consider the domain description D_{door} of Ex. 3.2. D_{door} is complete since it has only one model:*

$$\begin{aligned}
\sigma_0 &= \langle \{On\}, \{Closed\}\rangle \\
\Phi(Open,\sigma) &= \langle \sigma^+ \setminus \{Closed\}, \sigma^- \cup \{Closed\}\rangle \\
\Phi(Close,\sigma) &= \langle \sigma^+ \cup \{Closed\}, \sigma^- \setminus \{Closed\}\rangle \\
\Phi(Switch, \langle \sigma^+ \cup \{On\}, \sigma^- \setminus \{On\}\rangle) &= \langle \sigma^+ \setminus \{On\}, \sigma^- \cup \{On\}\rangle \\
\Phi(Switch, \langle \sigma^+ \setminus \{On\}, \sigma^- \cup \{On\}\rangle) &= \langle \sigma^+ \cup \{On\}, \sigma^- \setminus \{On\}\rangle \\
\Phi(\{Open, Close\},\sigma) &= \langle \sigma^+ \setminus \{Closed\}, \sigma^- \setminus \{Closed\}\rangle \\
\Phi(\{Open, Close, Switch\},\sigma) &= \langle \Phi^+(Switch,\sigma) \setminus \{Closed\}, \Phi^+(Switch,\sigma) \setminus \{Closed\}\rangle \\
\Phi(\{Open, Switch\},\sigma) &= \langle \Phi^+(Open,\sigma) \cup \Phi^+(Switch,\sigma), \\
&\qquad \Phi^-(Open,\sigma) \cup \Phi^-(Switch,\sigma), \rangle \\
\Phi(\{Close, Switch\},\sigma) &= \langle \Phi^+(Close,\sigma) \cup \Phi^+(Switch,\sigma), \\
&\qquad \Phi^-(Close,\sigma) \cup \Phi^-(Switch,\sigma)\rangle
\end{aligned}$$

Example 3.10 *Consider the domain description D_{soup} of Ex. 3.3. D_{soup} is complete since it has only one model:*

$$\begin{aligned}
\sigma_0 &= \langle \emptyset, \{Spilled\}\rangle \\
\Phi(Lift_Left,\sigma) &= \langle \{Spilled\}, \emptyset\rangle \\
\Phi(Lift_Right,\sigma) &= \langle \{Spilled\}, \emptyset\rangle \\
\Phi(\{Lift_Left, Lift_Right\},\sigma) &= \langle \emptyset, \{Spilled\}\rangle
\end{aligned}$$

3.3 Remarks

Now consider a domain description:

$$\begin{array}{c} \textbf{initially} \;\; \neg F \\ A \;\textbf{causes}\;\; F \;\textbf{if}\;\; G \\ A \;\textbf{causes}\;\; F \;\textbf{if}\;\; \neg G \end{array}$$

At the first sight, F would be true after A is done in the initial situation, as it seems that the effect of A has nothing to do with G, i.e. no matter whether G is initially true or false, F would always be true after A is done. According to our semnatics, however, we cannot derive it. Is there something wrong with our semantics? In what follows we will argue and justify our semantics.

Recall that our semantics is based on three-valued truth assignments to fluents in which the excluded middle law is not accepted. In the interpretation structure a *state* σ is a pair of sets of fluent names $\langle\sigma^+, \sigma^-\rangle$ such that σ^+ and σ^- are disjoint, i.e., $\sigma^+ \cap \sigma^- = \emptyset$. The reason that F would be true after A is done in the initial situation seems to be as follows: either G is initially true or G is initially false. In either case, by the two effect proposition we would have F being true after A is done. This reasoning is essentially based on the use of the excluded middle law, which is a moot or philosophical point regarding negation. Practical experience in logic programming for AI has shown the need for considering many different forms of negation [3, 23, 52, 71]. Our stance has been a cautious one, which relies on a skeptical semantics that can be made stronger if desired, whereas the reverse is impossible to do.

When a user describes an action domain, what he/she describes can only be understood epistemically. That is to say: what can be predicted or explained, the two major purposes of an action domain description, are only the user's beliefs. There is a lot of evidence that one cannot describe an action domain which is always correct, in the sense that it always agrees with what turns out to be true in our common-sense reality. For example, consider an effect proposition:

$$\mathit{Shoot} \;\textbf{causes}\;\; \neg \mathit{Alive} \;\textbf{if}\;\; \mathit{Loaded}$$

looks very simple. Is it always correct? What if the bullet is a soft plastic one? What if the gun is a toy? What if Fred is dressed in a bullet-proof coat? What if Fred turns out to be the "Robocop"? In our domain description, each effect proposition had better be understood as a "rule" which is used to derive new beliefs from old beliefs. The new beliefs depend on beliefs of the **if**–list. Fluents which do not appear in the **if**–list are regarded as irrelevant.

In our work, we defined the semantics of effect propositions in terms of states and transitions so that explicit modal operators are not involved and it becomes close to, though not the same as, the original proposal of A. In the semantics, the resulting state $\Phi(A, \sigma)$ denotes what one believes to be true or false. At this point we have only made a very weak assumption: what is believed to be true is disjoint from what is believed to be false (σ^+ and σ^- are always disjoint for any state σ). But if a fluent is not included in either of σ^+ and σ^-, its truth is regarded as unknown or undecided (we don't believe it is true, neither do we believe it is false). Although every fluent should have one and only one truth value in a situation, we may not know it. Actually, not only fluents but also mathematical propositions may have an unknown status of their truth values. That is the reason we reject the excluded middle law. The excluded middle principle is not accepted by all logicians, viz intuitionists, and so before being an AI and LP problem it is a logical one. For instance, when predications are made about inexistant referents: "The man on the moon is either old or not old" or when inappropriate predications are made: "The chair is either happy or not happy", etc.

If one accepts that a state is understood epistemically, there won't be any difficulty with the epistemical understanding of effect propositions. When we write

$$A \;\textbf{causes}\;\; F \;\textbf{if}\;\; G$$

we mean that if you believe G is true now, you believe F will be true after you do the action A. That is to say, the belief in F in the new situation depends on the belief in G in the current situation. If F is irrelevant to G epistemically, then we should write

$$A \textbf{ causes } F$$

We do not find people using common sense talking this way:

> If the President of the United States is Bill Clinton, the door will not be closed after you open the door.

Although the above effect proposition happens to be true now, people never talk this way when using common sense, as the status of the door has nothing to do with the President of the United States.

Now we return to the example presented in the beginning:

$$\textbf{initially } \neg F$$
$$A \textbf{ causes } F \textbf{ if } G$$
$$A \textbf{ causes } F \textbf{ if } \neg G$$

The two effect propositions means that F will be believed to be true after A is done in the situation where G is believed to be true or is believed to be false. That is to say, the belief in F should be supported by the belief in G or $\neg G$. If there is no evidence towards G or $\neg G$, F won't be believed as a result of performing A. Note that when one writes the above two effect propositions, he/she believes that G is relevant to F when A is done.

Now we use an instance of the above example to illustrate our semantics. Imagine a case in the following domain: a personnel manager evaluates three application packages about $John$, Tom and $Mary$. First, according to some rules, he/she gives some points based on information provided with the application package. It may happen that the manager can't give points for various reasons (the application form may not be legible, etc.). Suppose that there are only two possible points, say 0 and 1, and 0 denotes that the candidate is qualified and 1 denotes that the candidate is not qualified. After the manager gives the points, he/she will organize his/her archives, send some letters to inform applicants about being hired/rejected and thus close some application cases, and may plan to interview those applicants whose points have not been given either 0 or 1 and thus leaves the application case open for the moment.

Now we simply use two actions for the above domain: $Give\_Points$, $Organize\_Archive$. In the case of John, the manager happens to be unable to give points (John's handwriting is not legible). Then the user of the domain description is justified to say that he/she does not believe that the application case is closed after the actions $Give\_Points; Organize\_Archive$ are done in succession. The whole domain description, where fluents are self-explanatory, looks like this:

$$\textbf{initially } \neg Closed(John)$$
$$\textbf{initially } \neg Closed(Mary)$$
$$\textbf{initially } \neg Closed(Tom)$$
$$Give\_Points \textbf{ causes } Qualified(Mary)$$
$$Give\_Points \textbf{ causes } \neg Qualified(Tom)$$
$$Organize\_Archive \textbf{ causes } Closed(John) \textbf{ if } Qualified(John)$$
$$Organize\_Archive \textbf{ causes } Closed(John) \textbf{ if } \neg Qualified(John)$$
$$Organize\_Archive \textbf{ causes } Closed(Mary) \textbf{ if } Qualified(Mary)$$
$$Organize\_Archive \textbf{ causes } Closed(Mary) \textbf{ if } \neg Qualified(Mary)$$
$$Organize\_Archive \textbf{ causes } Closed(Tom) \textbf{ if } Qualified(Tom)$$
$$Organize\_Archive \textbf{ causes } Closed(Tom) \textbf{ if } \neg Qualified(Tom)$$

In our semantics we will have that the application cases for $Mary$ and Tom will be closed after $Give\_Points; Organize\_Archive$ are done in succession, while the status of the application case for John is left unchanged as a result of another assumption about the inertia law.

We should emphasize that the research into the semantics of negations in Logic Programming and Artificial Intelligence is an active topic and there have been a few proposals and uses of negations, i.e. classical negation, strong negation, explicit negation, constructive negation, inexact negation, etc. Our use of negation is similar to strong negation and explicit negation, as we only require that affirmative beliefs and negative beliefs be disjoint.

We observe that the example above only illustrates that the following two domain descriptions are not equal:

$$\begin{array}{c} \textbf{initially}\ \neg F \\ A\ \textbf{causes}\ \ F\ \textbf{if}\ \ G \\ A\ \textbf{causes}\ \ F\ \textbf{if}\ \ \neg G \end{array}$$

and

$$\begin{array}{c} \textbf{initially}\ \neg F \\ A\ \textbf{causes}\ \ F \end{array}$$

This observation signifies that the user of the tool provided in our paper should be cautious when he/she describes his/her domain. As we said before, if F has no relevance for G epistemically, then G should not be written in the **if**–list. This turns out to be a matter of usage of our tool. An incorrect use of our tool may lead to undesirable consequences, but it does not mean that our tool is wrong by itself. If F has nothing to do with G, why should we write

$$\begin{array}{c} A\ \textbf{causes}\ \ F\ \textbf{if}\ \ G \\ A\ \textbf{causes}\ \ F\ \textbf{if}\ \ \neg G \end{array}$$

instead of

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

Our paper provides a tool, which can certainly be mis-interpreted or misunderstood or misused. It may take time for the user to learn to use our tool, but an incorrect use of our tool leads to undesirable results does not mean that our tool is incorrect. This is also the case in other fields. For example, the logic program[6]

$$\begin{array}{rcl} holds(f, do(a, s)) & \leftarrow & holds(g, s) \\ holds(f, do(a, s)) & \leftarrow & \neg holds(g, s) \end{array}$$

is not equivalent to

$$holds(f, do(a, s)) \leftarrow$$

in a number of competing semantics of extended logic programs. Extended logic programs may be misused, but it does not mean these semantics of extended logic programs are wrong. What is wrong is that the extended logic programs are not used properly. When one wants to mean

$$holds(f, do(a, s)) \leftarrow$$

he/she can't write

$$\begin{array}{rcl} holds(f, do(a, s)) & \leftarrow & holds(g, s) \\ holds(f, do(a, s)) & \leftarrow & \neg holds(g, s) \end{array}$$

If an incorrect use of the semantics of extended logic programs leads to undesirable results, it does not mean that the semantics of extended logic programs is incorrect. Since the semantics of negations in extended logic programs do relate to common-sense reasoning, the discussions about them shouldn't be ignored in the discussions on negation in common sense reasoning about actions.

However, one may find it necessary in some domains for some fluents to be either true or false. If one wants to impose excluded middle law on some particular fluents, state constraints may be introduced to enhance $\mathcal{A}_{\mathcal{CO}}$. This is actually a current work topic of ours.

[6] Generally, the logic program $\{p \leftarrow q.\quad p \leftarrow \neg q.\}$ has a different semantics as the logic program $\{p \leftarrow .\}$, where $\neg$ denotes strong or explicit negation. See, for example, [3, 23, 52, 71] for some interesting discussions.

3.4 Comparison with related semantics

In the definition of the semantics of concurrent actions we have employed the inheritance principle: A concurrent action usually inherits effects of its subactions unless stated otherwise. As an example, consider the SCP domain, which is slightly different from D_{scp}:

$$\begin{array}{l} \textbf{initially } \neg Stolen \\ Steal \textbf{ causes } Stolen \\ \{Steal, Wait\} \textbf{ causes } Stolen \end{array}$$

The effect of concurrent action $\{Steal, Wait\}$ is the same as that of $Steal$. In other words, the effect of $Steal$ is inherited by $\{Steal, Wait\}$. By our definition we really have $\Phi(\{Steal, Wait\}, \sigma) = \Phi(Steal, \sigma)$.

The effect relationship between an action and its subactions was first discussed by Gelfond, Lifschitz and Robinov in the framework of the situation calculus [25], and further explored by Lin and Shoham [45], and Baral and Gelfond [7], respectively.

In [45], Lin and Shoham identified a problem called the *action-oriented frame problem*. The traditional frame problem is called the *fluent-oriented frame problem*. Their work is 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 [44] 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 [7], Baral and Gelfond made a first step to extend $\mathcal{A}$ 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: Usually a concurrent action inherits effects of its subactions (called inheritance postulate or inheritance principle later in this paper). It seems there are two differences between [45] and [7]. First, Baral and Gelfond used the negation-as-failure operator to formalize the inheritance postulate in extended logic programming with answer-set semantics [23], while Lin and Shoham used circumscription policy to deal with the action-oriented frame problem. Another difference, more important, is in their treatment of conflicting effects, as analyzed later.

Although the inheritance principle is adopted in this paper, our treatment is a little different from those of [45] and [7]. Lin and Shoham focused on epistemological completeness which means, intuitively, that a theory of a (deterministic) 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. Consider the concurrent action $\{Open, Close\}$ which tries to open and close a door at the same time. 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. 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.

Now let's see how Baral and Gelfond deal with the concurrent action $\{Open, Close\}$. In [7] the state transition is defined to be a partial mapping from sets of pairs (A, σ), where A is an action name and σ is a state, into sets of states. As a result, the state transition Φ is undefined in some states. Let $\sigma_0 = \langle\{Closed\}, \{On\}\rangle$. According to [7], indeed, $\Phi(\{Open, Close\}, \sigma_0)$ is undefined. Since $\Phi(\{Open, Close\}, \sigma_0)$ is undefined, the successive actions are not executable. Now we slightly modify the domain. Assume we have an additional action, $Switch$, which denotes switching a light, and an additional fluent On to denote that the light is bright. According to [7] the action sequence $\{Open, Close\}; Switch$ is unexecutable, which cannot be the case in the common-sense reality. For example, suppose that Mr Smith tries to open the door. Unfortunately, Mr Jones comes to make sure to close the door at exactly the same time as Mr Smith opens. It is then very controversial to say the door is open or closed now. However, no matter whether the door is open or closed, Mr Smith then switches on the light in the room. At this stage, at least we can say that it is bright in the room now. Obviously, Baral and Gelfond's semantics cannot handle this situation. As another example, consider a concurrent action $\{Open, Close, Switch\}$. Once again, Baral and Gelfond's state transition $\Phi(\{Open, Close, Switch\}, \sigma_0)$ is undefined. But in reality we still can at least assert that it is bright in the room in the resulting situation although we are not sure whether the door is open or closed.

Finally let's see how our semantics handle these examples. From our definition it can be seen that the state transition function is a total mapping, while the truth values of some fluents may be unknown. In our semantics, $\Phi(\{Open, Close\}, \sigma_0) = \langle\{\}, \{On\}\rangle$. This means that the status of the door is unknown after the action $Open$ and $Close$ are done at the same time, but the light is still off as before. And we also have $\Phi(\{Open, Close, Switch\}, \sigma_0) = \langle\{On\}, \{\}\rangle$, which implies that the light is bright after the three actions are done at the same time, but the status of the door is not known. In this respect we think that our treatment of concurrent actions has more natural results than Baral and Gelfond's.

As a final remark we observe that if an action A and an action B have a conflicting effect, for example, there are two e-propositions A **causes** F and B **causes** $\neg F$, all their effects are assumed unknown when they are done at the same time. For example, consider the following domain description:

$$
\begin{array}{l}
A \textbf{ causes } F \\
A \textbf{ causes } G \\
B \textbf{ causes } \neg F \\
C \textbf{ causes } H
\end{array}
$$

Let $\sigma_0 = \langle\{F, G\}, \{H\}\rangle$. Then, in its model (σ_0, Φ) the state transition Φ must satisfy $\Phi(\{A, B\}, \sigma_0) = \langle\{\}, \{H\}\rangle$, where G is also regarded as undefined. This might be too conservative. However, in absence of more evidence and justifications from our common-sense reality, we find that our conservative treatment would be a good choice, since it would assumedly not lead to absurd conclusions.

4 Abductive logic programming

Abduction is a procedure of synthetic inference starting from conclusions, whose task is as follows: Given a set of sentences T as theory, and a sentence G as observation, find a set of sentences Δ such that

- $T \bigcup \Delta \models G$;
- $T \bigcup \Delta$ is consistent;
- Δ is minimal in the sense that no proper subset Δ' of Δ exists such that Δ' satisfies the first two conditions.

The above third condition is not essential. Also, it is often presumed that by itself $T \not\models G$. The set of sentences Δ above acts as an explanation for observation G.

It has been recognized for several years that the logic programming paradigm can be extended to perform abductive reasoning [30]. In particular, abductive reasoning can be used for abductive model-based diagnosis, i.e., to abductively infer the causes of discrepancies between predicated and observed behaviour of a system. In Section 7 we will specially discuss temporal explanation from the stand-point of abductive model-based diagnosis.

As will be seen, the translations of domain descriptions in $\mathcal{A}_{\mathcal{CO}}$ will be acyclic, and thus their predicate completion models coincide with both the stable models [22] and the well-founded models [70]. In this paper we will follow Denecker [17] and use Console's predicate completion semantics by only completing defined predicates and leaving abducibles open.

A logic programming rule is of the form

$$A \leftarrow B_1, \ldots, B_m, not\ B_{m+1}, \ldots, not\ B_n$$

where A and each B_i are atoms, and not is the negation-as-failure operator.

An abductive logic programming framework is a triple (P, A, I), where P is a set of logic programming rules, A a set of abducibles, and I a set of integrity constraints. Sometimes abducibles are understood as predicate symbols and sometimes as literals (with or without NAF), and onstraints can be formulas in first-order or three-valued or modal logic, etc. In this paper we will take abducibles to be a set of predicate symbols, and constraints to be a set of first-order formulas.

As per Console [11], the semantics of a normal logic program can be defined through deduction by completing the non-abducible predicates together with Clark's Equality Theory (CET) [10].

Let P be a normal logic program, and A a set of abducibles. We write $COMP(P; A)$ to denote the first-order theory obtained by completing the non-abducible predicates of program P and by adding Clark's Equality Theory (CET). We will often omit CET, when no confusion will arise. A model of an abductive logic programming framework (P, A, I) is defined to be any first-order model of $COMP(P; A)$, which satisfies the integrity constraints I. Thus, the semantics of an abductive logic programming framework (P, A, I) can be defined to be the first-order theory $COMP(P; A) \cup I$. When there is no need mentioning A and I, we will simply write $COMP(P)$ to stand for $COMP(P; A) \cup I$. Let $\neg(C_1 \wedge \ldots \wedge C_m \wedge \neg C_{m+1} \wedge \ldots \wedge \neg C_n)$ be a first-order sentence, where each C_i is an atom. Then, it can be easily seen that it is equivalent to the completion of

$$false \leftarrow C_1, \ldots, C_m, not\ C_{m+1}, \ldots, not\ C_n$$

where $false$ is a special atom, always interpreted to be false in any first-order interpretation structure. Thus, if I is a set of formulas of the form $\neg(C_1 \wedge \ldots \wedge C_m \wedge \neg C_{m+1} \wedge \ldots \wedge \neg C_n)$, then an abductive logic programming framework (P, A, I) will often be represented as another abductive logic programming framework $(P', A, \emptyset)$ with empty integrity constraints, where P' is obtained from P and I. Lloyd and Topor [46] have developed a general technique which can transform a first-order formula into logic programming rules. This transformation is extended, and its correctness is proved by Denecker [16]. In what follows we will often write integrity constraints in the above form, if possible. An abductive answer to a query $? - Q$ in an abductive logic programming framework (P, A, I) is a set-inclusion minimal subset R of instances of A such that $COMP(P; A) \cup I \cup R$ is consistent and $COMP(P; A) \cup I \cup R \models Q$.

Example 4.1 *Consider an abductive logic programming framework $FOO = (P, A, I)$, where*

$$P = \left\{ \begin{array}{ccc} p & \leftarrow & a \\ p & \leftarrow & b \\ q & \leftarrow & b \end{array} \right\}, \quad A = \{a, b\}, \quad I = \{false \leftarrow p, q\}$$

Then, we have

$$COMP(P; A) \quad = \quad \left\{ \begin{array}{ccc} p & \leftrightarrow & a \vee b \\ q & \leftrightarrow & b \end{array} \right\}$$

The semantics of FOO is $COMP(P; A) \cup I$, which is equivalent to the following first-order theory

$$\{p \leftrightarrow a \vee b, q \leftrightarrow b, \neg(p \wedge q)\}$$

Suppose that we want to evaluate a query $? - p$. Then $\{a\}$ is an abductive answer, since it satisfies the following three conditions:

- $COMP(P; A) \cup I \cup \{a\} \models p$;
- $COMP(P; A) \cup I \cup \{a\}$ *is consistent.*

- *There is no proper subset of $\{a\}$ satisfying the above two conditions,and hence $\{a\}$ is a set-inclusion minimal subset satisfying the above two conditions.*

Note that $COMP(P; A) \cup I \cup \{b\}$ is not consistent. Thus, $\{b\}$ is not an abductive answer to $? - p$ in FOO. Now suppose that we want to evaluate a query $? - q$. It can be shown that there is no abductive answer to it.

In our translation we will only use acyclic logic programs[7] [4], which are a special class of normal logic programs with some nice properties. For acyclic logic programs, the predicate completion semantics coincides with other semantics such as the stable model semantics [22] and the well-founded model semantics [70]. In the case of abductive logic programs, the semantical coincidence still holds, as shown by Denecker [16].

There have been many proposals, e.g., [15, 65], for abductive query evaluation procedures for abductive normal logic programs. The work reported in this paper has been experimented with the latest version of the REVISE system [13], an extended logic programming system for revising knowledge bases. It is based on logic programming with explicit negation and integrity constraints and provides a two- or three-valued assumption revision to remove contradictions from the knowledge base. The latest version of the REVISE system is based on a top-down query evaluation procedure [1] of well founded semantics with explicit negation (WFSX) [52]. If explicit negation is not used, then the WFSX semantics coincides with the WFS semantics. Thus, the use of REVISE is not essential for the purpose of this paper, but it allows for experiment on extensions of $\mathcal{A}_{\mathcal{CO}}$ with more complicated constraints for ramifications which are not discussed in this paper.

One major use of the REVISE system is as follows: The REVISE system accepts an extended logic program P which may contain variables, a set of predicates or propositions as abducibles (called revisables in the REVISE system), and a set of integrity constraints expressed in the following form:

$$< -B_1, \ldots, B_2, not\ C_1, \ldots, not\ C_n$$

where B_i and C_j are literals (atoms or atoms preceded with explicit negation denoted by -). The head is understood as something equivalent to $false$ in this paper. The abducibles are asserted in one of the following forms:

$$\begin{array}{ll} :- & revisable(L) \\ :- & revisable(L, true) \\ :- & revisable(L, false) \end{array}$$

The predicate $revisable(L, false)$ is equivalent to $revisable(L)$. It declares L to be a revisable with an initial value $true$ or $false$ in the WFSX model, respectively. A solution R is defined to be a pair $[TrueLits, FalseLits]$, where $TrueLits$ and $FalseLits$ are two disjoint sets of literals, such that $P \cup \{L \leftarrow\ : L \in TrueLits\} \cup \{-L \leftarrow\ : L \in FalseLits\}$ is consistent. To obtain solutions from the REVISE system, the user just issues a query

$$? - solution(R)$$

The expected answer to it is of the form

$$R = [TrueLits, FalseLits]$$

where $TrueLits$ and $FalseLits$ are the revisable literals whose values should be changed to $true$ or $false$, respectively. $solution/1$ returns one solution at a time, non-deterministically.

For non-floundering program P, the REVISE system has the following properties:

[7] Let P be a normal logic program, and B_P the set of ground atoms of P. A *level mapping* for a program P is a function $\lambda : B_P \rightarrow N$ from ground atoms to natural numbers. For $A \in B_P$, $\lambda(A)$ is the *level* of A; Given a level mapping λ, we extend it to ground negative literals by letting $\lambda(\neg A) = \lambda(A)$; A clause of P is called *acyclic with respect to* a level mapping λ, if for every ground instance $A \leftarrow L_1, \ldots, L_n$ of it, we have $\lambda(A) > \lambda(L_i)$, for every $1 \leq i \leq n$. A program P is called em acyclic with respect to a level mapping λ, if all its clauses are. P is called *acyclic* if it is acyclic with respect to some level mapping.

- If the execution terminates with a failure on the query $?-solution(R)$, then there is no $R = [TrueLits, FalseLits]$ such that $P \cup \{L \leftarrow\ : L \in TrueLits\} \cup \{-L \leftarrow\ : L \in FalseLits\}$ is consistent.
- If the execution terminates and generates solutions $R_1, R_2, \ldots, R_m$, then for each other solution R, there exists a R_i which is more general than R in the sense that a substitution θ exists such that $\theta(R_i) \subseteq R$. If R_i and R are ground, this simply means that $R_i \subseteq R$.

For abductive purposes, the REVISE system can be used as follows: Let (P, A, I) be an abductive logic programming framework. First transform each constraint $C \in I$ into logic programming rules of the form :

$$false \leftarrow Body$$

Then, for each of them add the following to the program P to have a new program P':

$$< -\ Body$$

For each n-ary abducible predicate ab/n in A, add the following to P' to have a new program P'':

$$:-\ revisable(ab(\underbrace{\_, \ldots, \_}_{n})).$$

Suppose we want to evaluate a query $?-Q$. We just need to add the following to P'' to have P''':

$$< -not\ Q$$

which means that if Q is not true, then there is a contradiction in P'''. A solution $R = [TrueLits, FalseLits]$ for removing such a contradiction is an abductive answer to $?-Q$ in the sense that Q belongs to the WFSX model of $P'' \cup \{L \leftarrow\ : L \in TrueLits\} \cup \{-L \leftarrow\ : L \in FalseLits\}$. It can be seen that the REVISE can provide more than we need.

Example 4.2 *Consider the abductive logic programming framework FOO in the previous example. Create a file, say foo, which contains:*

```
p <- a.
p <- b.
q <- b.
<- p, q.
:- revisable(a).
:- revisable(b).
<-  not p.
```

The last rule above is intended for the query ? – p. After booting the REVISE system, we can read foo into the REVISE system and compute the abductive answers to $\leftarrow p$. The following is the running result:

```
| ?- read_file(foo).
yes
| ?- solution(R).
R = [[a],[]] ? ;
no
| ?-
```

It means that $\{a\}$ is an abductive answer to ? – p.

We should emphasize that the REVISE system is designed for extended logic programs with explicit negation. Since we will only use acyclic normal logic programs, other abductive query evaluation procedures for abductive normal logic programs can also be used.

5 From domain descriptions in $\mathcal{A}_{\mathcal{CO}}$ to abductive normal logic programs

In this section we will present a translation from domain descriptions in $\mathcal{A}_{\mathcal{CO}}$ to abductive logic programs. We will compare this translation with others in Section 8.

In the translation we will use some predicates whose informal meanings are as follows[8]:

Predicates or Functions	Meanings
$Initially\_true/1$	$Initially\_true(F)$ means that fluent F is initially true.
$Initially\_false/1$	$Initially\_true(F)$ means that fluent F is initially false.
$Is\_true/2$	$Is\_true(F,S)$ means that fluent F is true in situation S.
$Is\_false/2$	$Is\_false(F,S)$ means that fluent F is false in situation S.
$Initiates/3$	$Initiates(A,F,S)$ means that fluent F is initiated in situation S by action A.
$Terminates/3$	$Terminates(A,F,S)$ means that fluent F is terminated in situation S by action A.
$Result/2$	The function $Result(A,S)$ denotes the new situation by doing action A in situation S.
$Subacteq/2$	$Subacteq(A,B)$ denotes that A is a subaction of B.
$Subact/2$	$Subact(A,B)$ denotes that A is a proper subaction of B, that is, $Subacteq(A,B)$ but A is not B.
$Immediate\_Initiates/3$	$Immediate\_Initiates(A,F,S)$ means that fluent F is immediately initiated in situation S. This predicate will be used to translate effect propositions. Let A **causes** F **if** $p_1,\ldots,p_n$ be an e-proposition, where F is positive. It will be translated into something like: $Immediate\_Initiate(A,F,S)$ if every p_i holds in situation S.
$Immediate\_Terminates/3$	$Immediate\_Terminates(A,F,S)$ means that fluent F is immediately terminated in situation S. This predicate will be used to translate effect propositions. Let A **causes** $\neg F$ **if** $p_1,\ldots,p_n$ be an e-proposition, where F is positive. It will be translated into something like this: $Immediate\_Terminate(A,F,S)$ if every p_i holds in situation S.
$Causes/3$	$Causes(F,S,A,B)$ denotes B is the set-inclusion minimal subaction of A satisfying the following condition: F is immediately initiated (or terminated) by subaction B of A, and is not terminated (or initiated) by any other action C, which is not only a proper superaction of B but also a subaction of A.
$Delta/3$	$Delta(A,S,F)$ corresponds to something like $F \in \Delta(A,\sigma)$, where $\Delta(A,\sigma)$ is defined in Subsection 3.2.

[8] We often use the PROLOG notation $/n$ for n-ary predicates/functions. For example, we write $Is\_true/2$ to mean that Is_true is a binary predicate.

We will also use a few other secondary predicates: $Clip\_Initiates(F, B, A, S)$, $Clip\_Terminates(F, B, A, S)$, $Clip\_Cause1(F, B, A, S)$, $Clip\_Cause2(F, B, A, S)$, whose meanings will be informally given when they are defined.

Among the above predicates, the predicates $Initially\_true/1$ and $Initially\_false/1$ are abducibles. The variables start with capital letters, while constants start with small-case letters. For any positive fluent expression F, i.e., $F \in \Sigma_f$, we define $|F| = |\neg F| = F$. For each $f \in \Sigma_f$, we have two constants f and $\overline{f}$, where $\overline{f}$ is intended to denote the negative fluent $\neg f$. The following translation has been made as close as possible to the definition of the semantics of domain descriptions in $\mathcal{A}_{\mathcal{CO}}$ at the cost of elegancy. More elegant programs can be obtained by simple program transformations. Let D be a domain description in $\mathcal{A}_{\mathcal{CO}}$. The translation πD consists of a normal logic program P_D and a set of constraints IC_D, defined as follows:

1. Auxiliary predicates about subactions:

   Assume that we are given a set of standard rules for set-theoretical predicates $Member(A, S)$, $Subseteq(S_1, S_2)$to determine if an element belongs to a set, and to determine if a set is a subset of another. A set can be represented by a PROLOG-like list. In what follows we will use the following predicates: $Member(A, S)$ to determine whether A is a member of S, $Subacteq(S_1, S_2)$ to determine whether S_1 is a subaction of S_2, and $Subact(S_1, S_2)$ to determine whether S_1 is a proper subaction of S_2, i.e., S_1 is a subaction of S_2 but is not equal to S_2. $Member(A, S)$, $Subacteq(S_1, S_2)$ and $Subact(S_1, S_2)$ can be easily defined from the set-theoretical predicates. Note that $Subacteq(S_1, S_2)$ is not the same as $Subseteq(S_1, S_2)$ in the sense that the empty action is not allowed. The definitions of these predicates do not depend on domain descriptions.

2. Initialization:

$$
\begin{aligned}
Is\_true(F, s_0) &\leftarrow Initially\_true(F) && (3)\\
Is\_false(F, s_0) &\leftarrow Initially\_false(F) && (4)
\end{aligned}
$$

   where s_0 is a new symbol to denote the initial situation. The predicates $Initially\_true(F)$ and $Initially\_false(F)$ are taken to be abducible. 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

   The following are auxiliary predicates, whose syntax does not depend on domains, though their semantics does. These auxiliary predicates will be used to define the two main predicates $Is\_true(F, S)$ and $Is\_false(F, S)$, which indicate whether fluent F is true or false in situation S. The reason their semantics depends on domains is that their semantics depends on domain-specific predicates.

$$
\begin{aligned}
Initiates(A, F, S) &\leftarrow Immediate\_Initiates(A, F, S) && (5)\\
Initiates(A, F, S) &\leftarrow Subacteq(B, A), \\
&\quad\;\; Immediate\_Initiates(B, F, S), \\
&\quad\;\; not\ Clip\_Initiates(F, B, A, S) && (6)\\
Terminates(A, F, S) &\leftarrow Immediate\_Terminates(A, F, S) && (7)\\
Terminates(A, F, S) &\leftarrow Subacteq(B, A), \\
&\quad\;\; Immediate\_Terminates(B, F, S), \\
&\quad\;\; not\ Clip\_Terminates(F, B, A, S) && (8)\\
Causes(F, S, A, B) &\leftarrow Subacteq(B, A), \\
&\quad\;\; Immediate\_Initiates(B, F, S),
\end{aligned}
$$

$$
\begin{aligned}
& & & not\ Clip\_Initiates(F,B,A,S), && (9)\\
& & & not\ Clip\_Cause1(F,B,A,S) && (10)\\
Causes(F,S,A,B) & \leftarrow & & Subacteq(B,A), && \\
& & & Immediate\_Terminates(B,F,S), && \\
& & & not\ Clip\_Terminates(F,B,A,S), && (11)\\
& & & not\ Clip\_Cause2(F,B,A,S) && (12)\\
Delta(A,S,F) & \leftarrow & & Initiates(A,G,S), Terminates(A,G,S), && \\
& & & Causes(G,S,A,B), Initiates(B,F,S) && (13)\\
Delta(A,S,F) & \leftarrow & & Initiates(A,G,S), Terminates(A,G,S), && \\
& & & Causes(G,S,A,B), Terminates(B,F,S) && (14)
\end{aligned}
$$

The predicate $Clip\_Initiates(F,B,A,S)$ (or $Clip\_Terminates(F,B,A,S)$, resp.) means that there is a proper superaction C of B, which is in turn a subaction of A, such that C cancels the effect of B, i.e., C immediately terminates F (or C immediately initiates F, resp.). The predicates $Clip\_Cause1(F,B,A,S)$ and $Clip\_Cause2(F,B,A,S)$ mean that B cannot be a cause for the change of the truth values of fluent name F in the situation S when A is done, since there is a proper superaction C, which is in turn a subaction of A, such that C changes the truth values of F in the same way as B does. That is to say, B is not the set-inclusion minimal action. These four secondary predicates are defined as follows:

$$
\begin{aligned}
Clip\_Initiates(F,B,A,S) & \leftarrow & & Subact(B,C), Subacteq(C,A), && \\
& & & Immediate\_Terminates(C,F,S) && (15)\\
Clip\_Terminates(F,B,A,S) & \leftarrow & & Subact(B,C), Subacteq(C,A), && \\
& & & Immediate\_Initiates(C,F,S) && (16)\\
Clip\_Cause1(F,B,A,S) & \leftarrow & & Subact(B,C), Subacteq(C,A), && \\
& & & Immediate\_Initiates(C,F,S), && \\
& & & not\ Clip\_Initiates(F,C,A,S) && (17)\\
Clip\_Cause2(F,B,A,S) & \leftarrow & & Subact(B,C), Subacteq(C,A), && \\
& & & Immediate\_Terminates(C,F,S), && \\
& & & not\ Clip\_Terminates(F,C,A,S) && (18)
\end{aligned}
$$

Since some literals in $Causes/4$ and $Clip\_Cause1$ are the same, one can introduce an additional predicate to simplify the rules of $Causes/4$ and $Clip\_Cause1$. Analogously for $Causes/4$ and $Clip\_Cause2/4$.

4. Main Predicates

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

$$
\begin{aligned}
Is\_true(F, Result(A,S)) & \leftarrow & & Is\_true(F,S), && \\
& & & not\ Terminates(A,F,S), not\ Delta(A,S,F) && (19)\\
Is\_true(F, Result(A,S)) & \leftarrow & & Initiates(A,F,S), && \\
& & & not\ Terminates(A,F,S), not\ Delta(A,S,F) && (20)\\
Is\_false(F, Result(A,S)) & \leftarrow & & Is\_false(F,S), && \\
& & & not\ Initiates(A,F,S), not\ Delta(A,S,F) && (21)\\
Is\_false(F, Result(A,S)) & \leftarrow & & Terminates(A,F,S), && \\
& & & not\ Initiates(A,F,S), not\ Delta(A,S,F) && (22)\\
& & & && (23)
\end{aligned}
$$

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:

   $$Immediate\_Initiates(a, f, S) \leftarrow Holds(p_1, S), \ldots, Holds(p_n, S) \tag{24}$$

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

   $$Immediate\_Terminates(a, f, S) \leftarrow Holds(p_1, S), \ldots, Holds(p_n, S) \tag{25}$$

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

$$Holds(F, Result(A_1; \ldots; A_m, s_0)) \tag{26}$$

which can be transformed into the following equivalent logic programming rule:

$$false \leftarrow not\ Holds(F, Result(A_1; \ldots; A_m, s_0)) \tag{27}$$

Recall that $Holds(F, S)$ is an abbreviation; it stands for $Is\_true(F, S)$ when F is a positive fluent, and for $Is\_false(G, S)$ when $F \equiv \neg G$ with G being a positive fluent. We should emphasize that the above form of integrity constraints are supported in the REVISE system. For other abductive query evaluation procedures, e.g. SLDNFA, which do not directly support integrity constraints, see Denecker [16] for the method to treat integrity constraints.

What is more, we cannot abduce a fluent to be both true and false. For this purpose we add the following domain-independent constraint:

$$\neg(Is\_true(F, S_0) \wedge Is\_false(F, S_0)) \tag{28}$$

which is equivalent to the following logic programming rule:

$$false \leftarrow Is\_true(F, S_0), Is\_false(F, S_0) \tag{29}$$

Note that many rules and an integrity constraint in the translation are domain-independent. It thus suffices to give those domain-specific logic programming rules and integrity constraints for a given domain description.

Example 5.1 *Consider the domain description D_{door} of Example 3.2. The domain-specific logic programming rules and integrity constraints are as follows:*

$$\begin{array}{rcl}
Immediate_Terminates(Open, Closed, S) & \leftarrow & \\
Immediate_Initiates(Close, Closed, S) & \leftarrow & \\
Immediate_Initiates(Switch, On, S) & \leftarrow & Is_false(On, S) \\
Immediate_Terminates(Switch, On, S) & \leftarrow & Is_true(On, S) \\
false & \leftarrow & not\ Is_false(Closed, s0) \\
false & \leftarrow & not\ Is_true(On, s0)
\end{array}$$

Proposition 5.2 *πD is an acyclic program with first-order constraints.*

Proof It suffices to construct a level mapping λ from the ground atoms $B_{\pi D}$ to natural numbers. Note that the predicates *Subacteq*, *Subact*, $Initially_t rue$ and $Initially_f alse$ has nothing to do with domain descriptions. But semantics of *Subacteq* and *Subact* does depend on the alphabet of actions. However, their semantics is always uniquely decided. Hence, in the foregoing definition of the level mapping we will simply regard them as having level 0. Let W be a set, we write $|W|$ to denote the number of elements of W. Let $S = Result(A_m, \ldots, Result(A_2, Result(A_1, s_0)))$ be a term in πD, we define $|S| = (2^{|A_1|} + 2^{|A_2|} + \ldots + 2^{|A_m|})$. There are many level mappings for πD. It is straightforward to verify that λ defined below, for example, is a level mapping. For any fluent expression F, action expression A, we define a level mapping λ as follows:

$$\begin{aligned}
\lambda(Initially_true(F)) &= 0 \\
\lambda(Initially_false(F)) &= 0 \\
\lambda(Subacteq(A, B)) &= 0 \\
\lambda(Subact(A, B)) &= 0 \\
\lambda(Immediate_Initiates(A, F, S)) &= 10 * |S| + 7 \\
\lambda(Immediate_Terminates(A, F, S)) &= 10 * |S| + 7 \\
\lambda(Clip_Initiates(F, B, A, S)) &= 10 * |Result(A, S)| + 1 \\
\lambda(Clip_Terminates(F, B, A, S)) &= 10 * |Result(A, S)| + 1 \\
\lambda(Clip_Cause1(F, B, A, S)) &= 10 * |Result(A, S)| + 2 \\
\lambda(Clip_Cause2(F, B, A, S)) &= 10 * |Result(A, S)| + 2 \\
\lambda(Causes(F, S, A, B)) &= 10 * |Result(A, S)| + 3 \\
\lambda(Initiates(A, F, S)) &= 10 * |Result(A, S)| + 4 \\
\lambda(Terminates(A, F, S)) &= 10 * |Result(A, S)| + 4 \\
\lambda(Delta(A, S, F)) &= 10 * |Result(A, S)| + 5 \\
\lambda(Is_true(F, S)) &= 10 * |S| + 6 \\
\lambda(Is_false(F, S)) &= 10 * |S| + 6
\end{aligned}$$

□

6 Soundness and completeness

In this section we will prove the soundness and completeness of our translation π. In the following we always assume D is any arbitrary domain description in $\mathcal{A}_{\mathcal{CO}}$ and πD its translation. The completion semantics of πD is denoted by $COMP(\pi D)$. The language (symbols appearing in πD) is denoted by $\mathcal{L}_{\pi \mathcal{D}}$, or simply by $\mathcal{L}$.

An interpretation structure (U, M) for a first-order theory T consists of a universe U and a mapping M such that (i) for any constant v, $M(v) \in U$; (ii) for any n-ary function symbol f, $M(f)$ is an n-ary function on U; (iii) for any n-ary predicate P, $M(P)$ is an n-ary relation on U. When no confusion arises, we often write M for the interpretation structure (U, M). An assignment is a mapping from variables in T to universe U. Given an interpretation structure and an assignment, a first-order formula can be interpreted to be *true* or *false*, called *truth values*, as usual. The truth values of any first-order sentence (formula without free variables) does not depend on assignments. Given a first-order theory T, an interpretation structure is called its *model* if every formula of T is interpreted to be true. In the foregoing proof we will use the following slightly abused notations without confusion: Let M be any interpretation structure of $\mathcal{L}_{\pi \mathcal{D}}$.

- Let F be any sentence. We will write $M \models F$ to denote F is true in M.
- Let f and s be any two elements of the universe of M. Then, we write $M \models Is\_true(f, s)$ to denote $\langle f, s \rangle \in M(Is\_true)$, and $M \models Is\_false(f, s)$ to denote $\langle f, s \rangle \in M(Is\_false)$, where

$M(Is\_true)$ and $M(Is\_false)$ are two binary relations as the interpretation of the predicate Is_true and Is_false, respectively.

- Let f be a fluent symbol, and s any element of the universe of M, we write $M \models Is\_true(f,s)$ to denote $\langle M(f), s\rangle \in M(Is\_true)$, where $M(f)$ is the interpretation of the fluent symbol f. Similarly for Is_false.

Definition 6.1 *Let M be any interpretation of $\mathcal{L}_{\pi\mathcal{D}}$, and s any element of the universe of M. Define $state_M(s) = \langle state^+_M(s), state^-_M(s)\rangle$, where*

$$state^+_M(s) = \{f : f \text{ is a fluent symbol and } M \models Is\_true(f,s)\}$$
$$state^-_M(s) = \{f : f \text{ is a fluent symbol and } M \models Is\_false(f,s)\}$$

Notice that $state_M(s)$ corresponds to a state of a structure for domain description D. Hence we can define a transition function Φ on it.

Notice that the following predicates are defined in the translation πD: $Is\_true/2$, $Is\_false/2$, $Initiates/3$, $Terminates/3$, $Immediate\_Initiates/3$, $Immediate\_Terminates/3$, $Causes/3$, and $Delta/3$. In addition to the Clark Equality Theory CET, the semantics $COMP(\pi D)$ includes the following equivalences:

$$\forall F,T : Is\_true(F,T) \leftrightarrow E_1 \vee E_2 \vee E_3 \tag{30}$$

where

$$\begin{aligned}
E_1 &\equiv T = s_0 \wedge Initially_true(F)\\
E_2 &\equiv \exists A,S : T = Result(A,S) \wedge Is_true(F,S) \wedge\\
&\quad \neg Terminates(A,F,S) \wedge \neg Delta(A,S,F)\\
E_3 &\equiv \exists A,S : T = Result(A,S) \wedge Initiates(A,F,S) \wedge\\
&\quad \neg Terminates(A,F,S) \wedge \neg Delta(A,S,F)
\end{aligned}$$

Notice that there is a similarity between T and the state σ, between $Result$ and Φ, and between $Is\_true(F,T)$ and $F \in \sigma^+$, respectively. Similarly we have completed $Is\_false/2$ as follows:

$$\forall F,T : Is\_false(F,T) \leftrightarrow E_1 \vee E_2 \vee E_3 \tag{31}$$

where

$$\begin{aligned}
E_1 &\equiv T = s_0 \wedge Initially_false(F)\\
E_2 &\equiv \exists A,S : T = Result(A,S) \wedge Is_false(F,S) \wedge\\
&\quad \neg Initiates(A,F,S) \wedge \neg Delta(A,S,F)\\
E_3 &\equiv \exists A,S : T = Result(A,S) \wedge Terminates(A,F,S) \wedge\\
&\quad \neg Initiates(A,F,S) \wedge \neg Delta(A,S,F)
\end{aligned}$$

The completed predicates $Immediate\_Initiates/3$ and $Immediate\_Terminates/3$ are as follows:

$$\begin{aligned}
&Immediate_Initiates(A,F,T) \leftrightarrow\\
&\quad \bigvee(\exists S : A = a \wedge F = f \wedge T = Result(a,S) \wedge Holds(p_1,S) \wedge \ldots \wedge Holds(p_n,S))
\end{aligned} \tag{32}$$

such that there is a correspondence between the disjunct above and all the effect propositions a **causes** f **if** $p_1, \ldots, p_n$ with f being a fluent symbol.

$$\begin{aligned}
&Immediate_Terminates(A,F,T) \leftrightarrow\\
&\quad \bigvee(\exists S : A = a \wedge F = f \wedge T = Result(a,S) \wedge Holds(p_1,S) \wedge \ldots \wedge Holds(p_n,S))
\end{aligned} \tag{33}$$

such that there is a correspondence between the disjunct above and all the effect propositions a **causes** $\neg f$ **if** $p_1, \ldots, p_n$ with f being a fluent symbol.

Completion of other predicates is as follows:

$$
\begin{aligned}
&\forall A, F, S : Initiates(A, F, S) \leftrightarrow && (34)\\
&\quad Immediate\_Initiates(A, F, S) \vee (\exists B : Subacteq(B, A) \wedge \\
&\quad Immediate\_Initiates(B, F, S) \wedge \wedge \neg Clip\_Initiates(F, B, A, S)) \\
&\forall A, F, S : Terminates(A, F, S) \leftrightarrow && (35)\\
&\quad Immediate\_Terminates(A, F, S) \vee (\exists B : Subacteq(B, A) \wedge \\
&\quad Immediate\_Terminates(B, F, S) \wedge \neg Clip\_Terminates(F, B, A, S)) \\
&\forall F, S, A, B : Causes(F, S, A, B) \leftrightarrow Subacteq(B, A) \wedge && (36)\\
&\quad ((Immediate\_Initiates(B, F, S) \wedge \\
&\qquad \neg Clip\_Initiates(F, B, A, S) \wedge \neg Clip\_Cause1(F, B, A, S)) \vee \\
&\quad (Immediate\_Terminates(B, F, S) \wedge \\
&\qquad \neg Clip\_Terminates(F, B, A, S) \wedge \neg Clip\_Cause2(F, B, A, S))) \\
&\forall A, S, F : Delta(A, S, F) \leftrightarrow && (37)\\
&\quad \exists B, G : (Initiates(A, G, S) \wedge Terminates(A, G, S) \wedge Causes(G, S, A, B)) \wedge \\
&\quad (Initiates(B, F, S) \vee Terminates(B, F, S)) \\
&\forall F, B, A, S : Clip\_Initiates(F, B, A, S) \leftrightarrow && (38)\\
&\quad \exists C : Subact(B, C) \wedge Subacteq(C, A) \wedge Immediate\_Terminates(C, F, S) \\
&\forall F, B, A, S : Clip\_Terminates(F, B, A, S) \leftrightarrow && (39)\\
&\quad \exists C : Subact(B, C) \wedge Subacteq(C, A) \wedge Immediate\_Initiates(C, F, S) \\
&\forall F, B, A, S : Clip\_Cause1(F, B, A, S) \leftrightarrow && (40)\\
&\quad \exists C : Subact(B, C) \wedge Subacteq(C, A) \wedge \\
&\quad Immediate\_Initiates(C, F, S) \wedge \neg Clip\_Initiates(F, C, A, S) \\
&\forall F, B, A, S : Clip\_Cause2(F, B, A, S) \leftrightarrow && (41)\\
&\quad \exists C : Subact(B, C) \wedge Subacteq(C, A) \wedge \\
&\quad Immediate\_Terminates(C, F, S) \wedge \neg Clip\_Terminates(F, C, A, S)
\end{aligned}
$$

Note that the predicates $Clip\_Cause1(F, B, A, S)$ and $Clip\_Cause2(F, B, A, S)$ are used only in the definition of $Causes(F, S, A, B)$. Thus, their appearances in (36) can be regarded as abbreviations for the right-hand sides of (40) and (41), respectively. For simplicity, we will not explicitly define them in the model construction in the foregoing proof, since it suffices to give a correct definition for $Causes(F, S, A, B)$.

Lemma 6.2 *Let D be a consistent domain description, and $Q = F$* **after** *$A_1; \ldots; A_n$ any value proposition. Then, for any model (σ_0, Φ) of D, there is a model M of πD such that Q holds in (σ_0, Φ) iff $M \models \pi Q$, i.e, iff $M \models Is\_true(F, Result(A_n, \ldots, Result(A_1, s_0)))$ when F is a fluent symbol, and $M \models Is\_false(F, Result(A_n, \ldots, Result(A_1, s_0)))$ when $F = \neg G$ for a fluent symbol G.*

Proof Given a model (σ_0, Φ) of D, we want to construct a Herbrand model M of πD such that Q holds in (σ_0, Φ) iff $M \models \pi Q$. Let U be the Herbrand universe. We are particularly interested in two subsets of U, called *situations* and *actions*, respectively, defined as follows:
(i) all action expressions are actions;
(ii) s_0 is a situation;
(iii) $Result(a, s)$ is a situation whenever a is an action and s a situation.

Notice that there are some elements in U which are neither actions nor situations. For example, $Result(s_0, s_0)$ also appears in U, since U is the Herbrand universe. The basic idea of the following proof is as follows: We define M such that for each term $s = Result(a_1; \ldots; a_n, s_0)$, we have

$$state_M(s) = \Phi(a_1; \ldots; a_n, s_0)$$

Now we proceed to construct a Herbrand model M. Define $state_M(T)$ for any element $T \in U$ as follows:
(i) If $T = s_0$, then

$$state_M(s_0) = \sigma_0$$

(ii) If T is not a situation in U, then

$$state_M(T) = \langle \emptyset, \emptyset \rangle$$

(iii) If $T = Result(a, s)$ for action a and some situation s, then

$$state_M(Result(a, s)) = \Phi(a, state_M(s))$$

Let $state_M(T) = \langle \sigma^+, \sigma^- \rangle$. We write $state_M^+(T)$ and $state_M^-(T)$ to stand for σ^+ and σ^-, respectively. Note that $state_M(Result(a, s))$ is well-defined, since $state_M(s_0)$ is well-defined.

Recall terminologies in Subsection 3.2. In the following definition, A and B are actions, F is a fluent name, S is a situation. M is defined as follows:

$$\begin{aligned}
M(Is_true) &\equiv \{\langle f, s\rangle : f \in state_M^+(s) \text{ for all } f \text{ and } s \text{ of } U\ \} \\
M(Is_false) &\equiv \{\langle f, s\rangle : f \in state_M^-(s) \text{ for all } f \text{ and } s \text{ of } U\ \} \\
M(Immediate_Initiates) &\equiv \{\langle A, F, S\rangle : F \text{ is a fluent name,} \\
&\qquad \text{action } A \text{ immediately initiates } F \text{ in } state_M(s_0)\} \\
M(Immediate_Terminates) &\equiv \{\langle A, F, S\rangle : F \text{ is a fluent name,} \\
&\qquad \text{action } A \text{ immediately initiates } \neg F \text{ in } state_M(S)\} \\
M(Initiates) &\equiv \{\langle A, F, S\rangle : A \text{ is an action, } F \in Initiate(A, state_M(S))\} \\
M(Terminates) &\equiv \{\langle A, F, S\rangle : A \text{ is an action, } F \in Terminate(A, state_M(S))\} \\
M(Causes) &\equiv \{\langle F, S, A, B\rangle : B \in Cause(F, state_M(S), A)\} \\
M(Delta) &\equiv \{\langle A, S, F\rangle : F \in \Delta(A, state_M(S))\} \\
M(Clip_Initiates) &\equiv \{\langle F, B, A, S\rangle : \text{ there is a subaction } C \text{ of } A \\
&\quad \text{such that } B \text{ is a proper subaction of } C \text{ and} \\
&\quad C \text{ immediately initiates } \neg F \text{ in } state_M(S)\ \} \\
M(Clip_Terminates) &\equiv \{\langle F, B, A, S\rangle : \text{ there is a subaction } C \text{ of } A \\
&\quad \text{such that } B \text{ is a proper subaction of } C \text{ and} \\
&\quad C \text{ immediately initiates } F \text{ in } state_M(S)\ \} \\
M(Initially_true) &\equiv \sigma_0^+ \\
M(Initially_false) &\equiv \sigma_0^-
\end{aligned}$$

It can be seen that the above definition just mirrors the definitions in Subsection 3.2. Now we want to show

1. For any value proposition Q of D, Q holds in (σ_0, Φ) iff $M \models \pi Q$.
2. M is a model of IC_D.
3. M is a model of $COMP(\pi D)$.

To prove 1: Let Q be F **after** $a_1; \ldots; a_m$ with F being a fluent name. Then, we will have:

$$\begin{aligned}
Q \text{ holds in } (\sigma_0, \Phi) &\Leftrightarrow F \in \Phi(a_1; \ldots; a_m, \sigma_0)^+ \\
&\Leftrightarrow F \in state_M^+(Result(a_1; \ldots; a_m, s_0)) \\
&\Leftrightarrow \langle F, Result(a_1; \ldots; a_m, s_0)\rangle \in M(Is_true) \\
&\Leftrightarrow M \models Is_true(F, Result(a_1; \ldots; a_m, s_0)) \\
&\Leftrightarrow M \models \pi Q.
\end{aligned}$$

Let Q be $\neg F$ **after** $a_1; \ldots; a_m$ with F being a fluent name. Similarly, we can have:

$$\begin{aligned}
Q \text{ holds in } (\sigma_0, \Phi) &\Leftrightarrow M \models Is_false(F, Result(a_1; \ldots; a_m, s_0)) \\
&\Leftrightarrow M \models \pi Q.
\end{aligned}$$

To prove 2: Since every integrity constraint in IC_D is from a value proposition of D, by the result 1 we immediately have that M is a model of IC_D, since every value proposition of D holds in (σ_0, Φ).
To prove 3: The equivalences (40) and (41) are regarded as abbreviations used in (36). Thus it suffices to prove that all equivalences (30) – (39) are all satisfied by M. Proofs for (32) – (39) are straightforward, since they are just the mirroring definitions of relevant concepts in Subsection 3.2. Proof for (31) is analogous to that for (30). In what follows we will only give a proof for (30). To prove that (30) is satisfied in M, we need to check all its ground instances of (30). Let F and T be any two elements in U. We will analyze the following possible cases for T.

1. $T = s_0$.

   The completed definition (30) of $Is\_true/2$ collapses to

   $$Is\_true(F, s_0) \leftrightarrow Initially\_true(F)$$

   Since $state_M(s_0)$ is defined to be σ_0, the above equivalence is trivially satisfied in M.

2. T is not a situation.

   Notice that $state_M(T) = \langle \emptyset, \emptyset \rangle$. On the other hand, the right-hand side of (30) collapses to $false$. Thus, (30) is satisfied in M.

3. $T = Result(a, s)$ for action a and situation s. By definition of $state_M(Result(a, s))$, we have $state_M(Result(a, s)) = \Phi(a, state_M(s))$. Since (σ_0, Φ) is a model of D, then we have $\Phi(a, state_M(s)) = \langle \sigma^+, \sigma^- \rangle$, where

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

   Suppose $\langle F, Result(a, s)\rangle \in M(Is\_true)$. We want to prove that the right-hand side of (30) is satisfied in M. By definition of $M(Is\_true)$, we have $F \in state_M^+(Result(a, s))$, which implies that

   $$\begin{aligned}
   F &\in state_M^+(s) \cup Initiate(a, state_M(s)) \\
   F &\notin Terminate(a, state_M(s)) \\
   F &\notin \Delta(A, state_M(s))
   \end{aligned}$$

   Hence, the right-hand side of (30) is satisfied in M. Now suppose the right-hand side of (30) is satisfied in M. It is easy to see that the right-hand side of (30) is equivalent to

   $$(Is\_true(F, s) \vee Initiates(a, F, s)) \wedge \neg Terminates(a, F, s) \wedge \neg Delta(a, s, F)$$

   which implies

   $$F \in ((state_M^+(s) \cup Initiate(a, state_M(s))) \setminus Terminate(a, state_M(s))) \setminus \Delta(a, state_M(s))$$

Let $\Phi(a, state_M^-(s)) = \langle S^+, S^- \rangle$. Since (σ_0, Φ) is a model of D, $F \in S^+$. By definition of $M(Is\_true)$, we have $\langle F, Result(a, s) \rangle \in M(Is\_true)$. Thus, the left-hand side of (30) is satisfied in M.

In conclusion (30) is satisfied in M.

□

The above proof has been inspired by Denecker [16], but it is different from that in Denecker [16]. The main differences lie in the following facts:

- Concurrent actions are not considered in Denecker [16]. What is more, our translation π is very different from that of [16] . The common-sense inertia law is explicitly represented in [16] , while it, together with the inheritance postulate, is implicitly represented in our translation. In our translation the fluents are three-valued, though literals in logic programming rules are two-valued. In [16], both fluents and literals in logic programming rules are two-valued. Thus Denecker [16] cannot deal with conflicting effect propositions for (atomic) actions, and thus restricts the range of applicability of his method.
- Notice that domain descriptions are typed (actions and fluents), while the logic program is type-free. Given a domain description D in $\mathcal{A}$, in order to deal with the untypedness of a logic program, Denecker first extends it to another domain description D' by allowing each term $t \in U$ as a fluent symbol and as an action symbol. Then consider D'. Denecker's proof is certainly elegant for his translation from $\mathcal{A}$ to abductive normal logic programs. In this paper, if we directly followed Denecker's idea by extending D to D' by allowing more fluent symbols and action symbols, we could still prove the above lemma. But it would be complicated, since we would have to consider atomic actions and concurrent actions at the same time. Instead of solving this problem at the level of domain descriptions as Denecker does, we have kept D itself and coped with the untypedness of logic programs in the construction of the model. Whether or not our proof is elegant is left to the reader's judgement.

Theorem 6.3 (Soundness) *Let D be any domain description. For any value proposition Q, if $COMP(\pi D) \models \pi Q$, then D entails Q.*

Proof If D is inconsistent, then every value proposition is entailed by D. Now assume D is consistent. Since D is consistent, D has at least one model. Then, for any model (σ_0, Φ) of D, by Lemma 6.2 there is a model M of πD such that for any value proposition Q, Q holds in (σ_0, Φ) iff $M \models \pi Q$. If $\pi D \models \pi Q$, then for any model M of πD, $M \models \pi Q$. Thus we can conclude that if $\pi D \models \pi Q$, for any model (σ_0, Φ) of D, Q holds in (σ_0, Φ), and hence D entails Q. Note that by saying that Q holds in (σ_0, Φ) we mean $F \in S^+$ if F is positive, and $F \in S^-$ if F is negative, where $S = \Phi(A_1; \ldots; A_m, \sigma_0)$. □

We now turn to the completeness.

Theorem 6.4 (Completeness) *Let D be a domain description. For any value proposition Q, if D entails Q, then $COMP(\pi D) \models \pi Q$.*

Proof Assume that D entails Q, and M is any arbitrary model of πD. We want to prove that M is also a model of πQ. It suffices to construct a model $M_a = (\sigma_0, \Phi)$ of D such that for any value proposition Q, $M \models \pi Q$ iff Q holds in M_a. Notice that this immediately implies that all value propositions of D hold in M_a since M is a model of πQ for each value proposition Q of D. First we define $state_M(S)$, where S is of the form $Result(A_1; \ldots; A_m, s_0)$, as follows:

$$state_M(S) = \langle state_M^+(S), state_M^-(S) \rangle$$

where

$$\begin{aligned} state_M^+(S) &= \{F : F \in \Sigma_f, M \models Is\_true(F, S)\} \\ state_M^-(S) &= \{F : F \in \Sigma_f, M \models Is\_false(F, S)\} \end{aligned}$$

Intuitively, $state_M(S)$ denotes all (positive and negative) fluents which hold in the situation denoted by S in M. Now we define $M_a = (\sigma_0, \Phi)$ as follows:

- Φ is defined such that it satisfies the effect propositions of D. Notice that there is a unique transition function Φ.
- σ_0 is defined to be $state_M(s_0)$.

Now we need to prove that for each value proposition Q, $M \models \pi Q$ iff Q holds in M_a. That is, we need to prove that for each sequence of actions $a_1, \ldots, a_n$

$$\Phi(a_1; \ldots; a_n, s_0) = state_M(Result(a_1; \ldots; a_n, s_0)) \tag{42}$$

We prove it by induction on n. For $n = 0$, this is trivial, since σ_0 is defined to be $state_M(s_0)$ by the definition above. Now assume that the equality (42) holds for $n = k$ with $k \geq 0$, we want to prove that it also holds for $n = k + 1$.

By definition of $\Phi(a_1; \ldots; a_k; a_{k+1}, s_0)$ and induction hypothesis, we have the following equalities:

$$\begin{aligned} \Phi(a_1; \ldots; a_k; a_{k+1}, s_0) &= \Phi(a_{k+1}, \Phi(a_1; \ldots; a_k, s_0)) \\ &= \Phi(a_{k+1}, state_M(Result(a_1; \ldots; a_k, s_0))) \end{aligned}$$

Let $Result(a_1; \ldots; a_k, s_0) = s$. It suffices to prove

$$\Phi(a_{k+1}, state_M(s)) = state_M(Result(a_{k+1}, s))$$

Let $state_M(s) = \sigma$. By Subsection 3.2 we have

$$\Phi(a_{k+1}, \sigma) = \langle S^+, \quad S^- \rangle$$

where

$$\begin{aligned} S^+ &= ((\sigma^+ \cup Initiate(a_{k+1}, \sigma)) \setminus Terminate(a_{k+1}, \sigma)) \setminus \Delta(a_{k+1}, \sigma) \\ S^- &= ((\sigma^- \cup Terminate(a_{k+1}, \sigma)) \setminus Initiate(a_{k+1}, \sigma)) \setminus \Delta(A, \sigma) \end{aligned}$$

Comparing all completed predicates (30) – (39) with definitions in Subsection 3.2, we can see the following correspondences:

Definitions in Subsection 3.2		Completed Predicates in πD
$B \in Cause(F, state_M(S), A)$	$\Longleftrightarrow$	$Cause(F, S, A, B)$
$F \in \Delta(A, state_M(S))$	$\Longleftrightarrow$	$Delta(A, S, F)$
$F \in Initiate(A, state_M(S))$	$\Longleftrightarrow$	$Initiates(A, F, S)$
$F \in Terminate(A, state_M(S))$	$\Longleftrightarrow$	$Terminates(A, F, S)$
$F \in state^+_M(S)$	$\Longleftrightarrow$	$Is\_true(F, S)$
$F \in state^-_M(S)$	$\Longleftrightarrow$	$Is\_false(F, S)$

Since M is a model of πD, all completed predicates (30) – (39) are satisfied in M. In particular, (30) for Is_true and (31) for Is_false are satisfied in M. Then, we have

$$\begin{aligned} state^+_M(Result(a_{k+1}, s)) &= \{F : F \in \Sigma_f, M \models Is\_true(F, Result(a_{k+1}, s))\} \\ state^-_M(Result(a_{k+1}, s)) &= \{F : F \in \Sigma_f, M \models Is\_false(F, Result(a_{k+1}, s))\} \end{aligned}$$

Let $F \in \Sigma_f$ be any fluent symbol. By (30) and some set-theoretical equalities we have:

$$\begin{aligned} F \in state^+_M(Result(a_{k+1}, s)) &\Longleftrightarrow M \models Is\_true(F, Result(a_{k+1}, s)) \\ &\Longleftrightarrow M \models (Is\_true(F, s) \vee Initiates(A, F, s)) \wedge \\ &\qquad \neg Terminates(A, F, s) \wedge \neg Delta(A, s, F) \\ &\Longleftrightarrow (F \in \sigma^+ \vee F \in Initiate(A, \sigma)) \wedge \\ &\qquad F \notin Terminate(A, \sigma) \wedge F \notin Delta(A, \sigma) \\ &\Longleftrightarrow F \in S^+ \end{aligned}$$

Analogously, we have

$$F \in state^-_M(Result(a_{k+1}, s)) \Longleftrightarrow F \in S^-$$

Thus, we have

$$\Phi(a_1; \ldots; a_k; a_{k+1}, s_0) = state_M(Result(a_1; \ldots; a_k; a_{k+1}, s_0))$$

□

Notice that the above theorem also generalizes the result in Denecker [16], even if we only consider atomic actions. Denecker's translation is complete only for e-consistent domain descriptions [16]. The condition of being e-consistent is necessary for the completeness of Denecker's translation. When a domain description D is not e-consistent, no transition function exists, and thus has no models, which makes every value proposition entailed by D. On the other hand, Denecker's translation of D may be consistent. In our framework, however, fluents are three-valued. Assume that a domain description includes two e-propositions

$$\begin{array}{lll} a & \textbf{causes} & f \textbf{ if } p_1, \ldots, p_n \\ a & \textbf{causes} & \neg f \textbf{ if } p_1, \ldots, p_n \end{array}$$

Although it is not e-consistent, a state transition function still exists in our framework, by which all fluents initiated or terminated by a are regarded as undefined in the new situation $Result(a, s)$ if $p_1, \ldots, p_n$ hold in s.

7 Observations and explanations

At what follows we study, from the stand-point of model-based diagnosis, a class of problems related to observations and explanations, of which the Stolen Car Problem (SCP) is an instance. We will consider it in $\mathcal{A}_{CO}$ and in the abductive logic programs, respectively. In general the problem is as follows: Given a domain description D which describes the predicted behaviour of a domain of actions and changes, and an observation O which reports some fluents hold after some actions are observed to be performed, we want to abduce what other actions may have also happened during the course of occurrences of known actions, that would explain the observations.

There are two basic distinct approaches to diagnosis when a system behaves abnormally: the heuristic approach and the model-based approach. In the heuristic approach to diagnosis, human experience is programmed in a certain domain. The power and applicability certainly depends on human experience. In model-based approach to diagnosis, knowledge about the structure and behaviour, both normal and faulty, of systems being diagnosed is provided so that the normal behaviour of the systems being diagnosed can be predicted and compared with the observed behaviour. If there is a discrepancy between the predicted and observed behaviour, model-based diagnosis systems can determine, by abduction say, which components are malfunctioning. Obviously, the power and applicability only depends on descriptions of the structure and behaviour of systems being diagnosed and underlying diagnosing algorithms. In what follows we concentrate on model-based diagnosis [27].

There are two fundamentally different formalizations of model-based diagnosis: consistency-based diagnosis and abductive diagnosis.

- Consistency-Based Diagnosis. According to this approach [60, 14, 26], a system is defined to be a triple $(SD, COMPS, OBS)$, where SD, called system description, is a set of first-order sentences; $COMPS$, called the system components, is a finite set of constants; OBS, called observations, a set of first-order sentences. A diagnosis for the system is a subset $D \subset COMP$ such that $SD \cup OBS \cup \{AB(c) \ : \ c \in D\} \cup \{\neg AB(c) \ : \ c \in COMPS \setminus D\}$ is consistent.

- Abductive Diagnosis. A representative of this approach is the generalized set covering (GSC) model [58] of diagnosis. In GSC model, a diagnostic problem is defined to be a tuple

(D, M, C, M'), where D is a finite set of *disorders*, M a finite set of *manifestations*, $C \subseteq D \times M$ is meant to capture the notion of *causation*, and $M' \subseteq M$ is the set of manifestations which have been observed to occur in the current diagnostic setting. The problem of diagnosis is recast as the problem of finding a minimal cover for observed manifestations, that is, a minimal subset $D' \subseteq D$ that causes M'.

The two approaches seem to be fundamentally different. The consistency-based approach looks for a set of abnormality assumptions that are consistent with observations, while the abductive approach looks for a set of causes, or fault modes of components, that will imply the observations. However, their connections have been known and established. For example, Reiter [60] showed that the GSC model can be represented in the consistency-based framework, and Poole [56] showed how to go from an abstract problem to a formal representation of the problem for abductive and consistency-based diagnosis, respectively. Both types of diagnosis can be uniformly treated by abduction, when an appropriate system description is defined: in the consistency-based approach only abnormality of components is abducible; in abductive diagnosis, fault modes are also abducibles.

Although the theories and principles of diagnosis are developed for diagnosis of concrete devices, they can be applied to abstract problems such as domains of actions. The methodology of diagnosis can be applied to domains of actions in at least two ways: (i) for actions, we can introduce a binary predicate $AB/2$ to express the abnormality of an action in a situation: $AB(a, s)$ means that action a is abnormal in a situation s. When an action is abnormal, it does not have expected behaviour; (ii) We can regard observed value propositions as *manifestations* and the actions as *disorders*. When one observes some manifestations, some disorders may have occurred. Certainly, the notion of causality in a domain of actions is not so simple as in the GSC model that is typically used for medical diagnosis. In a domain of actions, the notion of causality will be captured by a domain description in $\mathcal{A}_{\mathcal{CO}}$.

In the rest of this section, we will pursue the GSC approach and develop a methodology which is suitable for domains of actions such as the Stolen Car Problem. Extensive discussions on various diagnosis problems in domains of actions are currently under investigation and will be reported elsewhere.

First we will discuss diagnostic explanation in $\mathcal{A}_{\mathcal{CO}}$. We will regard domain descriptions in $\mathcal{A}_{\mathcal{CO}}$ as something similar to the set of causalities in the GSC model, and introduce *observed value propositions* to express the *manifestations*. If the manifestations are not consistent with expected value propositions, then we need to find the hidden actions (disorders) which have also occurred during the course of known actions. The task of looking for hidden actions can be done in $\mathcal{A}_{\mathcal{CO}}$.

Then we will discuss diagnostic explanation in abductive logic programs. We will show how to translate the diagnostic explanation problem in $\mathcal{A}_{\mathcal{CO}}$ into abductive logic programming, where looking for hidden actions amounts to evaluating a query in abductive logic programming.

Finally we discuss the relation between diagnostic explanations at both levels, namely, in $\mathcal{A}_{\mathcal{CO}}$ and in abductive logic programs.

7.1 Diagnostic explanation in $\mathcal{A}_{\mathcal{CO}}$

Consider the following domain description for SCP:

$$\begin{array}{l} \textbf{initially} \ \neg Stolen \\ Stolen \ \textbf{after} \ Wait; Wait; Wait \\ Steal \ \textbf{causes} \ Stolen \end{array}$$

As discussed in Section 2, this domain description is inconsistent, and thus has no models. Our stance is that we should make a distinction between domain descriptions and domain observations. A domain description predicts how the domain will behave, while domain observation tells how the domain actually behaves. Whenever there is a discrepancy between them, a diagnosis problem arises (Fig. 1). The task of diagnosis is to find the cause for the behavioural discrepancy.

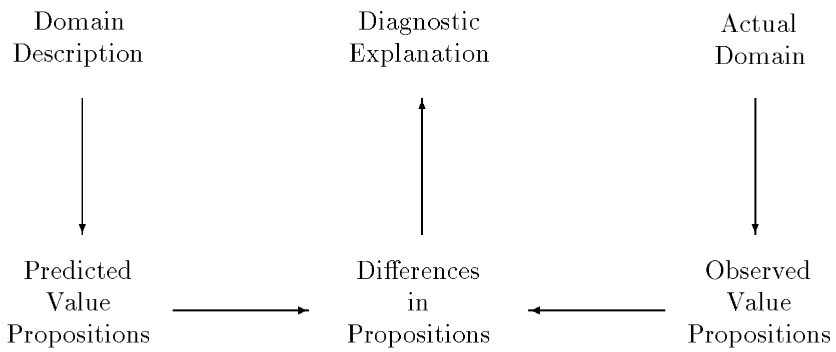

Figure 1: Diagnosis for Domains of Actions and Changes

In the Stolen Car Problem, the proposition

$$Stolen \ \textbf{after} \ \ Wait; Wait; Wait$$

should not be part of the domain description. Instead it is better regarded as an observed behaviour of the domain. Hence we write it as

$$\textbf{observed} \ \ Stolen \ \textbf{after} \ \ Wait; Wait; Wait$$

In general, where f is a fluent expression (a fluent symbol or its negation) and a_i is an action, the observed behaviour is written as:

$$\textbf{observed} \ \ f \ \textbf{after} \ \ a_1; a_2; \ldots; a_m \tag{43}$$

Statements of the form (43) are called *observation propositions*, or simply *o-propositions*. Notice that, syntactically, an observation proposition is closely related to a value proposition: If Q be a value proposition, then **observed** Q is an observation proposition, and vice versa. In the following, Q in **observed** Q will also simply be called an *observed value proposition*.

Definition 7.1 (Observation) *An observation OBS of a domain is a finite set of observation propositions of the form (43).*

For example, in the Stolen Car Problem domain, the observation OBS_{scp} is as follows:

$$\textbf{observed} \ \ Stolen \ \textbf{after} \ \ Wait; Wait; Wait$$

while its domain description D_{scp} for SCP is as follows:

$$\textbf{initially} \ \ \neg Stolen$$
$$Steal \ \textbf{causes} \ \ Stolen$$

Definition 7.2 (Diagnosis Problem) *Let D be a domain description in $\mathcal{A}_{CO}$, and $OBS = \{$ **observed** $Q_1, \ldots,$ **observed** $Q_m\}$ an observation. Let $OVP = \{Q_1, \ldots, Q_m\}$. We call (D, OBS) (or simply (D, OVP)) a diagnostic domain. We also say that there is a diagnostic explanation problem (or simply, diagnosis problem) for the domain iff $D \cup OVP$ is inconsistent.*

For example, there is a diagnostic explanation problem for SCP, since $D_{scp} \cup \{Stolen$ **after** $Wait$; $Wait$; $Wait\}$ is inconsistent.

Given a domain description D and an observation OBS, whether or not there is a diagnosis problem for (D, OBS) depends on whether or not a new domain description $D \cup OVP$ is consistent, where OVP is derived from the corresponding v-propositions of OBS. This implies that A_1 and A_2 do not necessarily occur concurrently in a diagnostic domain containing **observed** F **after** A_1 and **observed** G **after** A_2. In fact, the above definition mixes together several observations on different evolutions of a real domain. This enables us to refine v-propositions in a domain description step by step. A second possible choice is that we could regard A_1 and A_2 as occurring concurrently in a diagnostic domain containing **observed** F **after** A_1 and **observed** G **after** A_2. However, if we want to express that A_1 and A_2 occur concurrently in a diagnostic domain, we can write: **observed** F **after** $\{A_1, A_2\}$ and **observed** G **after** $\{A_1, A_2\}$. This means that our choice above does not lose generality with regard to the second choice.

As an example, let's consider a diagnosis domain (D, OBS), whose atomic action alphabet includes $\{a, b, c, d\}$ and fluent name alphabet includes $\{f, g\}$, where D includes

$$\begin{array}{l} \textbf{initially } f \\ \textbf{initially } \neg g \\ a \textbf{ causes } f \textbf{ if } g \\ b \textbf{ causes } g \\ c \textbf{ causes } \neg f \end{array}$$

and OBS includes

$$\begin{array}{l} \textbf{observed } f \textbf{ after } a \\ \textbf{observed } \neg f \textbf{ after } b \end{array}$$

Let OVP be

$$\begin{array}{l} f \textbf{ after } a \\ \neg f \textbf{ after } b \end{array}$$

Since for any state transition function Φ there is no state σ_0 such that all v-propositions in $D \cup OVP$ are satisfied in (σ_0, Φ), $D \cup OVP$ is inconsistent. Thus, there is a diagnosis problem for (D, OBS). Suppose we remove **observed** f **after** a from OBS to have, instead, an observation OBS'. There is still a diagnosis problem for (D, OBS').

Notice that there is always a diagnostic explanation problem for the domain if the domain description D itself is inconsistent. If D is inconsistent, we may think that the domain is not well-described. In this paper, we assume a domain description is well-described.

When there is a diagnostic explanation problem, we need to find the cause for the inconsistency of the diagnosis problem. Since we assume the domain description is correct and describes the predicted behaviour of the domain, the origin of the difference between predicted and observed behaviour is that either some qualifications of actions are violated, or some unobserved actions have happened. In this paper, we don't discuss qualifications. Thus, we assume that some unobserved actions have happened when the observed behaviour does not agree with the predicted behaviour.

Definition 7.3 (Expansion) *Let*

$$f_a \textbf{ after } a_1; \ldots; a_m \tag{44}$$

$$f_b \textbf{ after } b_1; \ldots; b_m \tag{45}$$

be two value propositions. The v-proposition (45) is said to be an expansion of (44) iff $f_a = f_b$ *and for every* $1 \leq i \leq m$, $a_i \subseteq b_i$. *Let* P_1 *and* P_2 *be two sets of value propositions.* P_2 *is said to be an expansion of* P_1 *iff (i) for every value proposition* Q_1 *of* P_1 *there is a value proposition* Q_2 *in* P_2 *which is an expansion of* Q_1*; and (ii) every value proposition* Q_2' *of* P_2 *is an expansion of a value proposition* Q_1' *of* P_1.

Definition 7.4 (Diagnostic explanation) *Given a diagnostic domain* (D, OVP)*, a diagnostic explanation for it is a set of value propositions* E *such that*

- *$D \cup E$ is consistent;*
- *E is an expansion of OBS.*

An immediate consequence of the above definition is the following corollary:

Corollary 7.5 *Suppose $D \cup OVP$ is consistent. Then OVP is a diagnostic explanation for (D, OVP).*

When $D \cup OVP$ is consistent, we needn't diagnose the action domain, since no contradiction arises. Furthermore, we can consider that the domain description D has been refined by $D \cup OVP$, which is then considered as a better domain description than D, when D is not complete.

For example, the following three sets are diagnostic explanations for the SCP diagnostic domain $D_{scp} \cup OBS_{scp}$:

$$\begin{aligned} E_1 &= \{Stolen \textbf{ after } \{Steal, Wait\}; Wait; Wait\} \\ E_2 &= \{Stolen \textbf{ after } Wait; \{Steal, Wait\}; Wait\} \\ E_3 &= \{Stolen \textbf{ after } Wait; Wait; \{Steal, Wait\}\} \end{aligned}$$

Any of E_1, E_2, E_3 will explain why the car is missing from the parking lot. Now suppose that we have an additional action symbol, say $Drink$. Then, the following set of value explanations is also an diagnostic explanation for $D_{scp} \cup OBS_{scp}$:

$$E_4 = \{Stolen \textbf{ after } \{Steal, Wait, Drink\}; Wait; Wait\}$$

Comparing E_4 with E_1, we may prefer E_1, since it provides a *parsimonious* explanation. Note that E_4 is an expansion of E_1.

Because our knowledge about the initial situation may be incomplete, there may be two different diagnostic explanations E_1 and E_2 arising out of two different possible initial situations. For example, consider the following domain description D:

$$\begin{aligned} &a \textbf{ causes } f \\ &b \textbf{ causes } g \textbf{ if } \neg g \end{aligned}$$

Suppose we have an observation $OBS = \{\textbf{ observed } g \textbf{ after } a\}$. Then we have two explanations:

$$\begin{aligned} E_1 &= \{g \textbf{ after } a\} \\ E_2 &= \{g \textbf{ after } \{a, b\}\} \end{aligned}$$

Although E_2 is an expansion of E_1, we may have no reason for preference of E_1 over E_2, since they arise out of two different initial situations: one includes $\{\textbf{ initially } g\}$ and the other includes $\{\textbf{ initially } \neg g\}$. For convenience, from now on we restrict ourselves to complete domain descriptions. Recall that a domain description is complete iff it has one and only one model. If one wants to extend our results to incomplete domain descriptions, all possible initial situations must be taken into account when defining explanations.

Definition 7.6 (Preferred Explanation) *Suppose that E_1 and E_2 are two diagnostic explanations for a given diagnostic domain (D, OVP). We say that E_1 is preferred to E_2 iff E_2 is an expansion of E_1. A diagnostic explanation for a diagnostic domain (D, OVP) is said to be most preferred iff for any diagnostic explanation E' for (D, OVP) if E' is preferred to E, then $E = E'$.*

For example, E_1 is a preferred explanation to E_4 for SCP diagnostic domain $D_{scp} \cup OBS_{scp}$. Moreover, all of E_1, E_2 and E_3 are most preferred diagnostic explanations.

Notice that the number of most preferred diagnostic explanations may decrease with more observed value propositions. For example, for the SCP domain description D_{scp}, suppose that we have the following observed value propositions:

$$\textbf{observed}\ \neg Stolen\ \textbf{after}\ \ Wait; Wait$$
$$\textbf{observed}\ \ Stolen\ \textbf{after}\ \ Wait; Wait; Wait$$

Then there is only one most preferred diagnostic explanations, i.e. E_3:

$$E_3 \quad = \quad \{Stolen\ \textbf{after}\ \ Wait; Wait; \{Steal, Wait\}\}$$

It is of practical interest how to find new observations in order to look for the real causes for the discrepancy between the predicted and observed value propositions.

7.2 Diagnostic explanation in abductive logic programs

Now we want to compute preferred diagnostic explanations by means of abductive logic programs. First of all, we have to translate observation propositions into rules of abductive logic programs. Although an observation proposition consists of a value proposition, the translation is not the same. The reason is that when we say something like **observed** $Stolen$ **after** $Wait; Wait; Wait$, we do not exclude the possibility of occurrences of other actions in the course of $Wait; Wait; Wait$. Since the term $Result(A, S)$ denotes the situation which is obtained by doing action A in the situation S, we cannot simply translate **observed** $Stolen$ **after** $Wait; Wait; Wait$ into something like $Is\_true(Stolen, Result(Wait; Wait; Wait, s_0))$. Suppose we know the action A is being done now, we cannot use $Happens(A, S)$ to denote it as in the event calculus [35]. The reason is simply as follows: In the action language $\mathcal{A_{CO}}$, the underlying time structure is branching, while in the event calculus the time structure is essentially linear. In order to keep the branching time structure, we use $OCC(A, S_1, S_2)$ to denote that the action A occurs in situation S_1 and leads to the situation S_2 possibly together with some other actions. For diagnostic purpose we introduce another abducible predicate $Happens/3$. The atom $OCC(A, S_1, S_2)$ and $Happens(A, S_1, S2)$ have the same meaning and could be denoted by only one predicate. The reason that we have deliberately introduced such predicates is that we want to make abducible predicates explicit. The predicate $Happens/3$ is an abducible one.

Let OBS be a set of observations, we write πOBS to denote its translation into abductive logic programs. In πOBS we will need many new symbols to represent situations, defined as follows: If $a_1; a_2; \ldots; a_n$ is a sequence of action expressions, then $s_{a_1;a_2;\ldots;a_n}$ is a new symbol. πOBS consists of a set of logic programming rules P_{OBS} and a set of integrity constraints IC_{OBS}, which are defined as follows:

1. For each o-proposition **observed** f **after** $a_1; a_2; \ldots; a_m$, we have the following m logic programming rules:

$$OCC(a_1, s_0, s_{a_1}) \leftarrow$$
$$OCC(a_2, s_{a_1}, s_{a_1;a_2}) \leftarrow$$
$$\ldots$$
$$OCC(a_m, s_{a_1;a_2;\ldots;a_{m-1}}, s_{a_1;a_2;\ldots;a_m}) \leftarrow \tag{46}$$

   And adding the following into IC_{OBS} as a constraint:

$$false \leftarrow not\ Holds(f, s_{a_1;a_2;\ldots;a_m}) \tag{47}$$

2. We add the following additional rules for $OCC/3$:

$$OCC(A, S_1, S_2) \quad \leftarrow \quad Atomic(A), Happens(A, S_1, S_2) \tag{48}$$

where $Atomic(A)$ denotes that A is an atomic action, i.e. $A \in \sigma_a$.

For efficiency of search in the implementation, we can introduce a new unary predicate, say $Sit/1$, to denote the situation constants s_0 and all the new symbols of the form $s_{a_1;a_2;\ldots;a_i}$ introduced in the previous step. Then, add the following new rules:

$$Sit(s_{a_1}) \leftarrow$$
$$\ldots$$
$$Sit(s_{a_1;a_2;\ldots;a_m}) \leftarrow$$

and change the above rule (48) into the following rule:

$$\begin{aligned} OCC(A, S_1, S_2) \quad \leftarrow \quad & Atomic(A), Sit(S_1), Sit(S_2), \\ & Happens(A, S_1, S_2) \end{aligned} \tag{49}$$

Still for efficiency of search, we even add more constraints in the body of the rule (49) to implement the so-called preference strategies often used in diagnosis.

3. We add the following into IC_{OBS} as a constraint[9]:

$$\begin{aligned} S_2 = Result(A, S_1) \quad \leftrightarrow \quad & \forall a \in A : OCC(a, S_1, S_2), \\ & \forall b \notin A : \neg OCC(b, S_1, S_2) \end{aligned} \tag{50}$$

Then, for a given diagnostic domain (D, OBS), its translation is defined to be $\pi D \cup \pi OBS$. Note that the logic programming rule (48) and constraint (50) are domain independent. Only rules in the first step above are domain specific. For example, the domain-specific rules and constraint in πOBS_{scp} for the SCP diagnostic domain are as follows:
Program P:

$$\begin{aligned} Atomic(Wait) \quad & \leftarrow \\ Atomic(Steal) \quad & \leftarrow \\ OCC(Wait, s_0, s_{Wait}) \quad & \leftarrow \\ OCC(Wait, s_{Wait}, s_{Wait;Wait}) \quad & \leftarrow \\ OCC(Wait, s_{Wait;Wait}, s_{Wait;Wait;Wait}) \quad & \leftarrow \\ OCC(A, S_1, S_2) \quad & \leftarrow \quad Atomic(A), Happens(A, S_1, S_2) \\ false \quad & \leftarrow \quad not\ Is\_true(Stolen, s_3) \end{aligned}$$

It can be shown that $COMP(P) \cup IC \cup \{Happens(Steal, s_0, s_1)\}$ is consistent, and the set of abduced atoms $\{Happens(Steal, s_0, s_1)\}$ can be obtained by, for example, running the REVISE system after the integrity constraint (50) is transformed into normal logic programming rules with help of the techniques discussed by Lloyd and Topor [46]. Actually, $\{Happens(Steal, s_0, s_1)\}$ corresponds to the diagnostic explanation:

$$E_1 \quad = \quad \{Stolen\ \textbf{after}\ \ \{Steal, Wait\}; Wait; Wait\}$$

With respect to $\pi D \cup \pi OBS$, it can be shown that it is still an acyclic program. To see why, notice that the new rules are only about $OCC/3$, whose bodies are either empty or abducibles ($Happens/3$), and hence the level mapping for πD can be trivially extended for $\pi D \cup \pi OBS$.

[9] Logic programming rules characterizing properties of $=$ are also needed. Since there are only a few predicates in our translation, in our practical implementation we just add rules for each predicate and function for the substitution property of $=$. For example, for substitution in Is_true we have a rule:

$$Is\_true(F, S_2) \leftarrow S_1 = S_2, Is\_true(F, S_1)$$

We will not deviate into more discussion on practical implementations.

Proposition 7.7 *Given a diagnostic domain (D, OBS), $\pi D \cup \pi OBS$ is an acyclic normal logic program with first-order constraints.*

Since $\pi D \cup \pi OBS$ is an acyclic abductive normal logic program with first-order constraints, we will still define its semantics as the first-order theory obtained by completing the defined predicates, together with first-order constraints as before. Thus, we can use an abductive query procedure to generate abductive answers to queries. In particular, an abductive answer, if any, may be generated for the query $?-not\ false$. If such an answer Δ exists, the theory $COMP(P_D \cup P_{OBS}) \cup IC_D \cup IC_{OBS} \cup \Delta$ is consistent. In the REVISE system, in order to compute the abductive answer to the query $?-not\ false$ we just need to issue the command $:-solution(R)$.

7.3 Soundness and completeness of explanations

We have introduced diagnostic explanation in $\mathcal{A}_{\mathcal{CO}}$, and we have also indicated that an abductive answer to the query $?-not\ false$ can be generated from the underlying abductive procedure. Our translation π is said to be sound in diagnostic explanation if for every abductive answer to $?-not\ false$ there is a diagnostic explanation, and π is said to be complete in diagnostic explanation if for every diagnostic explanation there is an abductive answer to $?-not\ false$.

Notation Let A and S be two finite sets of symbols. Let Δ be any subset of $\{Happens(a, s_1, s_2) : a \in A, s_1, s_2 \in S\}$. Then, we write $H(\Delta, s_1, s_2)$ to denote $\{a : Happens(a, s_1, s_2) \in \Delta\}$.

In the following we will also write Σ_s to denote all the new symbols introduced in predicate $OCC(A, s_i, s_j)$ of the translation $\pi D \cup \pi OBS$. Obviously, $s_0 \in \Sigma_s$. Given a query $?-not\ false$, we may have an abductive answer Δ to it. Notice that there are three abducible predicates: $Initially\_true/1$, $Initially\_false/1$ and $Happens/3$. We will thus partition Δ into Δ_I and Δ_H such that Δ_I only contains $Initially\_true/1$ and $Initially\_false/1$, and Δ_H only contains $Happens/3$, respectively. In what follows by saying that $\pi D \cup \pi OBS \cup \Delta_H$ is consistent we mean that there is an abductive answer Δ_I to $?-not\ false$, i.e. $\pi D \cup \pi OBS \cup \Delta_H$ has at least one model. Since we have restricted ourselves to complete domain descriptions, there is one and only one model (σ_0, Φ) and Δ_I is uniquely determined by σ_0 (See the proof of Lemma 6.2).

Theorem 7.8 (Explanation Soundness) *Let (D, OBS) be a diagnostic domain. If there exists Δ_H such that $\pi D \cup \pi OBS \cup \Delta_H$ is consistent, then there is a diagnostic explanation E for (D, OBS).*

Proof It suffices to construct such a diagnostic explanation E for (D, OBS). Let $\Sigma_s = \{s_0, s_1, \ldots, s_m\}$ be all the symbols introduced in the translation π. Let $\Delta_H = \{Happens(a_i, s_j, s_k) :$ for some action symbols a_i, and $s_j, s_k \in \Sigma_s\}$ be such that $\pi D \cup \pi OBS \cup \Delta_H$ is consistent. Define E as follows:
1. For each **observed** f **after** $a_1; \ldots; a_m \in OBS$, let $s_1, \ldots, s_m$ be the symbols introduced for $OCC(a_i, s_{i-1}, s_i)$, then

$$f \textbf{ after } a_1 \cup H(\Delta_H, s_0, s_1); \ldots; a_m \cup H(\Delta_H, s_{m-1}, s_m) \in E$$

2. No other value propositions belong to E.
Then, it can be shown that E is a diagnostic explanation. □

Theorem 7.9 (Explanation Completeness) *Let (D, OBS) be a diagnostic domain. If there is a diagnostic explanation for (D, OBS), then there is a Δ_H such that $\pi D \cup \pi OBS \cup \Delta_H$ is consistent.*

Proof It suffices to construct a Δ_H such that $\pi D \cup \pi OBS \cup \Delta_H$ is consistent. Define Δ_H as follows:
1. Let f **after** $a_1; \ldots; a_m \in E$ be an expansion of f **after** $b_1; \ldots; b_m \in OVP$, where OVP corresponds to OBS. Then, for every $c \in b_i \setminus a_i$, $Happens(c, s_j, s_k) \in \Delta_H$, where s_j and s_k are symbols introduced in $OCC(b_i, s_j, s_k)$;
2. No other elements belong to Δ_H.
The rest is straightforward. □

Corollary 7.10 *If $D \cup OVP$ is consistent, then there is an abductive answer Δ to ? – not false such that no atoms of the form $Happens(a, s_i, s_j)$ appear in Δ.*

8 Related work

There have been many reports on temporal reasoning. It is extremely difficult, if not impossible, to compare our work with all others. But there are still a few reports closely related to our own work. With respect to 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 compare our work with Gelfond and Lifschitz' original approach [24], Denecker's approach [17, 16], and Baral and Gelfond's approach [7]. We will also briefly remark on Dung's [18] at the end of Subsection 8.2.

8.1 Gelfond and Lifschitz' original approach

In [24] Gelfond and Lifschitz first proposed the action language $\mathcal{A}$, presented a transformation π_{GL} from domain descriptions in $\mathcal{A}$ into extended logic programs with answer sets as semantics.

We will write $\overline{L}$ to denote the literal complementary to L. Let D be any domain description in $\mathcal{A}$. The translation $\pi_{GL}D$ includes four rules for the law of inertia:

$$\begin{aligned}
Holds(F, Result(A, S)) &\leftarrow Holds(F, S), not\ Noninertial(F, A, S)\\
\neg Holds(F, Result(A, S)) &\leftarrow \neg Holds(F, S), not\ Noninertial(F, A, S)\\
Holds(F, S) &\leftarrow Holds(F, Result(A, S)), not\ Noninertial(F, A, S)\\
\neg Holds(F, S) &\leftarrow \neg Holds(F, Result(A, S)), not\ Noninertial(F, A, S)
\end{aligned}$$

Each value proposition f **after** $a_1; \ldots; a_m \in D$ is translated into

$$Holds(f, Result(a_1; \ldots; a_m, s_0))$$

Each effect proposition a **causes** f **if** $p_1, \ldots, p_m \in D$ is translated into $2m + 2$ rules:

$$\begin{aligned}
Hold(f, Result(a, S)) &\leftarrow Holds(p_1, S), \ldots, Holds(p_m, S)\\
Noninertial(|f|, a, S) &\leftarrow not\ \overline{Holds(p_1, S)}, \ldots, not\ \overline{Holds(p_m, S)}
\end{aligned}$$

And for each $i, i \leq i \leq m$:

$$\begin{aligned}
Hold(p_i, S) &\leftarrow \overline{Holds(f, S)}, Hold(f, Result(a, S))\\
\overline{Holds(p_i, S)} &\leftarrow \overline{Holds(f, Result(a, S))},\\
&\qquad Holds(p_1, S), \ldots, Holds(p_{i-1}, S),\\
&\qquad Holds(p_i, S), \ldots, Holds(p_m, S)
\end{aligned}$$

Gelfond and Lifschitz also showed that their translation π_{GL} is sound for domain descriptions without similar effect propositions, but not complete. The main differences between [24] and ours are as follows:

- Concurrent actions are not directly supported in $\mathcal{A}$, while they are in $\mathcal{A}_{\mathcal{CO}}$.
- In our framework fluents are three-valued. It seems more natural to us to regard fluents as undefined (their truth values are unknown) if they are both initiated and terminated at the same time. For example, assume D includes three effect propositions a **causes** f, a **causes** $\neg f$, b **causes** g. It seems that g should be true after a and b are done in sequence from the beginning, although we have no way to determine whether f is true or not in absence of further information.

- Domain descriptions and observations can't be distinguished in $\mathcal{A}$. It is no surprise that some domain descriptions such as SCP have no models. On the other hand, a distinction between domain descriptions and observations is made, and can be specified by different propositions in $\mathcal{A}_{\mathcal{CO}}$. According to the stance of $\mathcal{A}_{\mathcal{CO}}$, the domain description of a domain should only contain the propositions which are used to tell the predicted behaviour of the domain, while the domain observation contains actual value propositions which might be in disagreement with predicted value propositions.

- 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 Console predicate completion semantics.

8.2 Denecker's approach

In [17] Denecker elaborated on Gelfond and Lifschitz' original work, and proposed a translation π_D from domain descriptions in $\mathcal{A}$ to abductive normal logic programs. Actually, our current translation has been inspired by Denecker's work. However, some limitations and deficiencies of [24] are also inherited in [17]. The main differences between our work and Denecker's are summarized as follows:

- As in [24], concurrent actions are not directly supported in $\mathcal{A}$, while they are in $\mathcal{A}_{\mathcal{CO}}$.

- Domain descriptions and observations can't be distinguished in $\mathcal{A}$. Thus, some domain descriptions such as SCP have no models in Denecker [17]. But we have made a distinction between domain descriptions and observations. Thus, we can abductively generate explanations for domain observations given domain descriptions.

- Fluents are three-valued in our framework, but they are two-valued in $\mathcal{A}$.

- Our translation is complete for any domain description, but Denecker's translation is complete only for e-consistent domain descriptions.

Another approach, similar to Denecker's [17], was proposed by Dung [18]. In [18] Dung also elaborated on Gelfond and Lifschitz [24] and proposed a translation from $\mathcal{A}$ to logic programs with a special semantics similar to the completion semantics. In particular, Dung considered its application in database integrity constraints in the presence of updates. Concurrent actions and the distinction between domain descriptions and observations are not considered in Dung [18].

8.3 Baral and Gelfond's approach

In [7], Baral and Gelfond made a first step to extend $\mathcal{A}$ with concurrent actions. The new language is denoted by $\mathcal{A}_{\mathcal{BG}}$ in this paper. The syntax of $\mathcal{A}_{\mathcal{BG}}$ is the same as that of $\mathcal{A}$ except that an action name in $\mathcal{A}_{\mathcal{BG}}$ is defined to be a finite set of atomic action symbols.

The semantics is defined as follows. As defined in [24], 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* is a mapping Φ_{BG} of a subset of the set of pairs (A, σ), where A is an action name and σ is a state, into the set of states. Thus, the transition function Φ_{BG} is actually a partial mapping; this is different from that in [24] and in ours. As in [24], a *structure* is defined to be a pair (σ_0, Φ_{BG}), where σ_0 is a state (the *initial state* of the structure), and Φ_{BG} is a transition function. A sequence of action names $a_1, \ldots, a_m$ is said to be *executable* in a structure $M = (\sigma_0, \Phi_{BG})$ if for every $1 \leq k \leq m$ $\Phi_{BG}(a_k, \Phi_{BG}(a_{k-1}, \ldots, \Phi_{BG}(a_1, \sigma_0) \ldots))$ is defined.

A v-proposition f **after** $a_1; \ldots; a_m$ is *true* (*false*) in a structure M if $a_1, \ldots, a_m$ is executable in M and f holds (does not hold) in $M^{(a_1, \ldots, a_m)}(= \Phi_{BG}(a_m, \Phi_{BG}(a_{m-1}, \ldots, \Phi_{BG}(a_1, \sigma_0) \ldots)))$.

The execution of an action a in a state σ is said to *immediately cause* a fluent expression f if there is an e–proposition a **causes** f **if** $p_1, \ldots, p_n$ from the domain D such that for every i, $1 \leq i \leq n$, p_i holds in σ.

The execution of an action a in a state σ is said to *cause* a fluent expression f if a immediately causes f, or there is a $b \subseteq a$ such that the execution of b in σ immediately causes f and there is no c such that $b \subset c \subseteq a$ where the execution of c in σ causes $\neg f$.

Let A be an action and σ a state. Define

$$\begin{aligned} B_f(A,\sigma) &= \{f : f \text{ is a fluent name and execution of } A \text{ in } \sigma \text{ causes } f\} \\ B'_f(A,\sigma) &= \{f : f \text{ is a fluent name and execution of } A \text{ in } \sigma \text{ causes } \neg f\} \end{aligned}$$

A structure (σ_0, Φ_{BG}) is called a *model* of a domain description D if the following conditions are satisfied:

- Every v-proposition from D is true in (σ_0, Φ_{BG})
- For every action $A = \{a_1, \ldots, a_n\}$ and every state σ
  - if $B_f(A,\sigma) \cap B'_f(A,\sigma) = \emptyset$ then $\Phi_{BG}(A,\sigma)$ is defined and $\Phi_{BG}(A,\sigma) = (\sigma \cup B_f(A,\sigma)) \setminus B'_f(A,\sigma)$.
  - otherwise $\Phi_{BG}(A,\sigma)$ is undefined.

If a v-proposition Q is true in all models of a domain description D, D is said to *entail* Q.

Baral and Gelfond presented two translations in [7, 8]. In [8] disjunctive logic programs are used, which is another paradigm. In the rest of this paper we will only discuss the translation π_{BG} from $\mathcal{A}_{\mathcal{BG}}$ to extended logic programs proposed in [7]. Let D be a domain description in $\mathcal{A}_{\mathcal{BG}}$. The program $\pi_{BG}D$ consists of the translations of the individual propositions from D along with other axioms. To be specific, $\pi_{BG}D$ includes four rules for the law of inertia[10]:

$$\begin{aligned} Holds(F, Result(A,S)) &\leftarrow Holds(F,S), Atomic(A), \\ &\quad\; not\ Noninertial(F,A,S) \\ \neg Holds(F, Result(A,S)) &\leftarrow \neg Holds(F,S), Atomic(A), \\ &\quad\; not\ Noninertial(F,A,S) \\ Holds(F,S) &\leftarrow Holds(F, Result(A,S)), Atomic(A), \\ &\quad\; not\ Noninertial(F,A,S) \\ \neg Holds(F,S) &\leftarrow \neg Holds(F, Result(A,S)), Atomic(A), \\ &\quad\; not\ Noninertial(F,A,S) \end{aligned}$$

Each value proposition f **after** $a_1; \ldots; a_m \in D$ is translated into

$$Holds(f, Result(a_1; \ldots; a_m, s_0))$$

Each effect proposition a **causes** f **if** $p_1, \ldots, p_m \in D$ is translated into following $2m+2$ rules:

$$\begin{aligned} Hold(f, Result(a,S)) &\leftarrow Holds(p_1,S), \ldots, Holds(p_m,S) \\ Noninertial(|f|, a, S) &\leftarrow not\ \overline{Holds(p_1,S)}, \ldots, not\ \overline{Holds(p_m,S)} \end{aligned}$$

And for each $i, i \leq i \leq m$:

$$\begin{aligned} Hold(p_i,S) &\leftarrow \overline{Holds(f,S)}, Hold(f, Result(a,S)) \\ \overline{Holds(p_i,S)} &\leftarrow \overline{Holds(f, Result(a,S))}, \\ &\quad\; Holds(p_1,S), \ldots, Holds(p_{i-1},S), \\ &\quad\; Holds(p_i,S), \ldots, Holds(p_m,S) \end{aligned}$$

[10] Notice the difference between the law of inertia in π_{BG} and that in π_{GL}. The law of inertia in π_{BG} only applies to atomic actions.

The rules above differ from those in [24] only by allowing terms for compound actions. The next rules are new. They describe how the effects of individual actions are related to the effects of these actions performed concurrently.

$$\begin{array}{llcl}
(a) & Holds(F, Result(A,S)) & \leftarrow & subsetof(B,A), Holds(F, Result(B,S)), \\
 & & & not\ Noninherit(F,A,B,S) \\
(b) & \neg Holds(F, Result(A,S)) & \leftarrow & subsetof(B,A), \neg Holds(F, Result(B,S)), \\
 & & & not\ Noninherit(\overline{F},A,B,S)
\end{array}$$

And for each effect proposition a **causes** f **if** $p_1, \ldots, p_m \in D$, there is a rule

$$\begin{array}{lcl}
Noninherit(\overline{f},X,Y,S) & \leftarrow & subsetof(Y,X), subsetof(a,X), \neg subsetof(a,Y), \\
 & & not\ \overline{Holds(p_1,S)}, \cdots, not\ \overline{Holds(p_m,S)}
\end{array}$$

The soundness of π_{BG} is also indicated in [7]: For any v-proposition $Q = f$ **after** $a_1; \ldots, a_n$ and arbitrary domain description D such that $a_1, \ldots, a_n$ is executable in any model of D then if $\pi_{BG}D$ entails $\pi_{BG}Q$, then D entails Q.

Now we want to compare our π with π_{BG}. Notice that at the syntax level a domain description in $\mathcal{A}_{\mathcal{BG}}$ is also a domain description in $\mathcal{A}_{\mathcal{CO}}$, and the inheritance postulate is adopted both in $\mathcal{A}_{\mathcal{BG}}$ and in $\mathcal{A}_{\mathcal{CO}}$. But there are still some differences, as summarized as follows:

- Baral and Gelfond define the state transition as partial mapping functions, while we define the state transition as total mapping functions. For example, assume a domain description includes two effect propositions *Open* **causes** $\neg Closed$ and *Close* **causes** *Closed*. Then, the situation resulting from doing $\{Open, Close\}$ in any situation is not defined in $\mathcal{A}_{\mathcal{BG}}$, and thus all successive actions are disabled and are regarded as unexecutable.

- Although the inheritance postulate is adopted both in $\mathcal{A}_{\mathcal{BG}}$ and in $\mathcal{A}_{\mathcal{CO}}$, there is a difference with respect to conflicting effects of subactions of a concurrent action, as discussed in Subsection 3.4.

- As in [24], domain descriptions and observations can't be distinguished in $\mathcal{A}_{\mathcal{BG}}$. As analyzed before, a domain description should describe the predicted behaviour of the domain, while observations describe the observed behaviour. As shown before, the Stolen Car Problem can be solved in our framework, but cannot be solved in [7].

- Our translation is complete for any domain description, but π_{BG} inherits the incompleteness of π_{GL}.

9 Summary and concluding remarks

In this paper we have examined the action description language $\mathcal{A}$, and analyzed why there are no models for some domains such as the Stolen Car Problem. We have extended $\mathcal{A}$ by adding concurrent actions with a semantics somewhat different from that of $\mathcal{A}_{\mathcal{BG}}$. We have also enriched the action description language $\mathcal{A}$ with observation propositions to describe actually observed behaviour of domains of actions. By using domain description propositions (value and effect propositions) and domain observation propositions we can make a distinction between the predicted behaviour and actually observed behaviour of domains of (concurrent) actions, without requiring that they be consistent with each other. We have presented a translation from domain descriptions and observations to abductive logic programs, proved the soundness and completeness of the translation, and compared our work with others. In particular, from the stand-point of diagnosis we have discussed the temporal explanation of inferring actions from fluent changes at two different levels, namely, at the domain description level and at the abductive logic programming level, respectively, and relate them to each other by the soundness and completeness theorems. Our method is applicable to the temporal projection problem with incomplete information, as well as to the temporal explanation of inferring actions from fluent changes. All examples and a few additional benchmark examples have been experimented with our REVISE system.

This paper is a continuing experiment on using logic programs for representing and reasoning about actions and changes in line with [24, 17, 18, 7, 8]. Undoubtedly the new action language $\mathcal{A}_{\mathcal{CO}}$ still has some restrictions and limitations. For example, continuous changes, durative actions, and state constraints are not considered. The limitations and enhancements of $\mathcal{A}_{\mathcal{CO}}$ remain to be analyzed and done in future. Currently, extensive and intensive discussions on various diagnosis problems with domains of actions are also under investigation. Related work on belief update in action domains is also in progress and is partially reported in [38]. In addition, we are also working on a multi-agent description language based on $\mathcal{A}_{\mathcal{CO}}$ for a group of autonomous hysteretic agents in a dynamic world.

Acknowledgements

This work was supported by the JNICT PROLOPPE project under PRAXIS/3/31/TIT/24/94, and a post-doctoral fellowship from JNICT under PRAXIS XXI/BPD/4165/94 to the first author. We would also like to thank numerous people and our colleagues who have read earlier versions of this paper and provided valuable comments which have lead to the current version. In particular, we would like to thank Vladimir Lifschitz for the debate and arguement on our use of the three-valued logic for the semantics of the action description language.

References

[1] Alferes, J. J. , Damásio, C. V., andPereira, L. M., SLX - A top-down derivation procedure for programs with explicit negation, In: M. Bruynooghe (ed.), *International Logic programming Symposium.* MIT Press, 1994

[2] Alferes, J.J., Damásio, C.V., and Pereira, L.M., A Logic Programming System for Non-Monotonic Reasoning, *Journal of Automated Reasoning,* (14):93-147, 1995

[3] Alferes, J.J., Pereira, L.M., Przymusinski, T. C., "Classical" negation in non-monotonic reasoning and logic programming, in: Henry Kautz and Bart Selman (eds.), Artificial Intelligence and Mathematics Workshop., Fort Lauderdale, 1996

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

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

[6] Baker, A.B., and Ginsberg, M.L., Temporal projection and explanation, *Proc. of the IJCAI89*, Morgan Kaufmann,1989, 906–911

[7] Baral, C. and Gelfond, M., Representing concurrent actions in extended logic programming, *IJCAI*, Morgan Kaufmann, 1993, pp. 866–871

[8] Baral, C. and Gelfond, M., *Reasoning about Effects of Concurrent Actions*, University of Texas at El Paso, Manuscript, 1994

[9] van Belleghem, K., Denecker, M., de Schreye, D., Combining situation calculus and event calculus, *Proc. of ICLP 95*, MIT Press, 1995

[10] Clark, K., Negation as failure, *Logic and Databases*, H. Gallaire and J. Minker (eds.), Plenum Press, 1978, 293–322

[11] 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

[12] Crawford, J. and Etherington, D.W., Formalizing reasoning about change: a qualitative reasoning approach, *Proc. of AAAI 92*, 1992, 577–583

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

[14] de Kleer, J., Mackworth, A. K., Reiter, R., Characterizing diagnoses and systems, *Artificial Intelligence*, 56, 1992, 197–222

[15] 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

[16] Denecker, M., *Knowledge Representation and Reasoning in Incomplete Logic Programming*, Ph.D. Thesis, Department of Computer Science, K.U.Leuven, 1993

[17] 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

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

[19] Eshghi, E. and Kowalski, R., Abduction compared with negation as failure, *Proc. of the 6th Int'l Conf. on Logic Programming*, G. Levi and M. Martelli eds., MIT Press, 1989, 234-254

[20] Evans, C., Negation-as-failure as an approach to the Hanks and McDermott problem, *Proc. of the 2nd Int't Symp. on Artificial Intelligence*, 1989

[21] Gelfond, M., Autoepistemic logic and formalization of commonsense reasoning, *Non-Monotonic Reasoning: Second International Workshop, Lecture Notes in Artificial Intelligence 346*, Springer-Verlag, 1989, 176-186

[22] Gelfond, M., and Lifschitz, V., The stable model semantics for logic programming, In. *Proc. of 5th Logic Programming Symposium*, R. Kowalski and K. Bowen, eds., MIT Press, 1988, 1070-1080

[23] Gelfond, M. and V. Lifschitz, Logic programs with classical negation, *Logic Programming: Proc. of the 7th Int'l Conf.*, D. Warren and P. Szeredi, (eds.), MIT Press, 1990, pp. 579-597

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

[25] 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

[26] Greiner, R., Smith, B., and Wilkerson, R., A correction to the algorithm in Reiter's theory of diagnosis, *Artificial Intelligence*, 41:1,1989, 79-88

[27] Hamscher, W., Console, L., and de Kleer, J. (eds.), *Readings in Model-Based Diagnosis*, Morgan Kaufmann, 1992

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

[29] Haugh, B.A., Simple causal minimizations for temporal persistence and projection, *Proc. of the AAAI87*, 1987, 218–223

[30] Kakas, A.C., Kowalski, R.A., Toni, F., Abductive logic programming, *J. of Logic and Computation*, 2;6, 1993, 719-770

[31] Kartha, G.N., Two counterexamples related to Baker's approach to the frame problem, *Artificial Intelligence* 69, 1994, 379–391

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

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

[34] Kautz, H.A., The logic of persistence, *Proc. of the AAAI86*, 1986, 401–405

[35] Kowalski, R.A. and Sergot, M., A logic-based calculus of events, *New Generation Computing*, 4, 1986, pp.67-75

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

[37] Li, R. and Pereira, L. M., Temporal reasoning with abductive logic programming, *Proc. of ECAI–96*, 1996

[38] Li, R. and Pereira, L. M., What is believed is what is explained (sometimes), *Proc. of AAAI–96*, 1996

[39] Lifschitz, V., Towards a metatheory of action, *Proc. of KR 91*, Morgan Kaufmann, 376–386

[40] Lifschitz, V., Formal theories of action, *The Frame Problem in Artificial Intelligence*, Morgan Kaufmann, 1987, 35–57

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

[42] Lifschitz, V. and Robinov, A., Miracles in formal theories of action, *Artificial Intelligence*, 38:2, 1989, 225–237

[43] Lifschitz, V., Pointwise circumscription, *Readings in Nonmonotonic Reasoning*, M.L.Ginsberg (ed.), Morgan Kaufmann, 1987, 410–423

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

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

[46] Lloyd, J.W. and Topor, R. W., Making prolog more expressive, *Journal of Logic Programming*, 1:3, 1984, 225-240

[47] McCarthy, J. and Hayes, P.J., Some philosophical problems from the stand-point of artificial intelligence, in: B. Meltzer and D. Michie, (eds.), *Machine Intelligence*, 4, Edinburgh, 1969, pp.463-502

[48] McCarthy, J., Applications of circumscription to formalizing common-sense knowledge, *Artificial Intelligence*, 28, 1986, 89-116

[49] Morris P., The Anomalous extension problem in default reasoning, *Artificial Intelligence*, 35(3), 1988, 383–399

[50] 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

[51] Pednault, E. P. D., ADL and the state-transition model, *Journal of Logic and Computation*, 4:5, 1994, 467–517

[52] Pereira, L. M. and Alferes, J. J., Well-founded semantics for logic programs with explicit negation, in *European Conf. on Artificial Intelligence*, B. Neumann (ed.), John Wiley & Sons, 1992, 102-106

[53] Pereira, L. M., Alferes, J. J., and Aparício, J. N., Nonmonotonic Reasoning with Well Founded Semantics, In: K. Furukawa (ed.), *Proc. of 8th ICLP* , MIT Press, 1991, 475-489

[54] Pereira, L. M., Aparício, J. N., and Alferes, J. J., Non–monotonic Reasoning with Logic Programming, *Journal of Logic Programming 17(2,3 & 4), Special issue on Nonmonotonic reasoning*, 1993, pp. 227–263

[55] Pinto, J. and Reiter, R., Temporal reasoning in logic programming: A case for the situation calculus, *Proc. of ICLP 93*, MIT Press, 203–221

[56] Poole, D., Representing diagnosis knowledge, *Annals of Mathematics and Artificial Intelligence*, 11, 1994, 33–50

[57] T. Przymusinski, On the declarative semantics of stratified deductive databases and logic programs, In J. Minker, editor, , Foundations of Deductive Databases and logic Programming, Morgan Kaufmann, 1987, 193-216

[58] Reggia, J.A., Nau, D.S., and Wang, Y., A formal model of diagnostic inference I: Problem formulation and decomposition, *Info. Sci* 37, 1985, 227–256

[59] Reiter, R., A logic for Default Reasoning, *Artificial Intelligence*, 13, 1980, pp. 81-132

[60] Reiter, R., A theory of diagnosis from first principles, *Artificial Intelligence*, 32:1,1987, 57-96

[61] 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

[62] Sandewall, E., Filter preferential entailment for the logic of action in almost continuous worlds, *Proc. of IJCAI'89*, Morgan Kaufmann, 1989

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

[64] Sandewall, E., The Range of Applicability of Some Non-monotonic Logics for Strict Inertia, *Journal of Logic and Computation*, Vol. 4, No. 5, 1994, pp. 581-615

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

[66] Shoham, Y., *Reasoning about Change*, The MIT Press, 1987

[67] Shanahan, M., Prediction is deduction but explanation is abduction, Proc. IJCAI89, Morgan Kaufmann, 1989, 1055–1061

[68] Shanahan, M., Explanation in the situation calculus, *Proc. of IJCAI 93*, Morgan Kaufmann, 1993, 160–165

[69] Stein, L.A., and Morgenstern, L., Motivated action theory: a formal theory of causal reasoning, *Artificial Intelligence*, 71, 1994, 1–42

[70] Van Gelder, A. , Ross, K. and Schlipf, J.S., The well-founded semantics for general logic programs, *J. ACM* 38 1991, 620-650

[71] Wagner, G., *Vivid Logic, Lecture Notes in Artificial Intelligence 764*, Springer Verlag, 1994