Computing Ramifications by Postprocessing

Michael Thielscher
Intellektik, Informatik, TH Darmstadt
Alexanderstraße 10, D–64283 Darmstadt (Germany)
mit@intellektik.informatik.th-darmstadt.de

Abstract

A solution to the ramification problem caused by underlying domain constraints in STRIPS-like approaches is presented. We introduce the notion of causal relationships which are used in a post-processing step after having applied an action description. Moreover, we show how the information needed for these post-computations can be automatically extracted from the domain constraints plus general knowledge of which fluents can possibly affect each other. We illustrate the necessity of causal relationships by an example that shows the limitedness of a common method to avoid unintended ramifications, namely, the distinction between so-called frame and non-frame fluents. Finally, we integrate our solution into a recently developed, STRIPS-like yet purely deductive approach to reasoning about actions based on Equational Logic Programming.

1 Introduction

The ramification problem [Finger, 1987] is usually regarded as one of the challenges to all formal frameworks for reasoning about actions and change. It states that it is unreasonable to specify explicitly all changes that are caused by the execution of some action. Rather the specification should concentrate on immediate effects, and so-called *indirect* effects are then implicitly derived by some additional general knowledge of dependencies. Consider, as an example, an electric circuit containing a switch and a light bulb. Here one might wish to specify that the only direct effect of toggling the switch is a change of its position while the state of the bulb is indirectly affected via a general law (called *domain constraint*) that describes the causal connection between the switch's state and the bulb [Lifschitz, 1990].

It has often been argued (see, e.g., [Ginsberg and Smith, 1988]) that STRIPS-like approaches [Fikes and Nilsson, 1971] are inherently unsuited to tackle the ramification problem since they are based on the idea that all fluents remain unchanged which are not explicitly mentioned in an action description. In this paper, however, we propose to regard the set of fluents which is obtained after having applied some action description merely as a preliminary approximation of the resulting situation—which then is computed by performing additional post-processing steps that model indirect effects according to given domain constraints.

Our approach consists of two steps. First, we show how the post-processing computations just mentioned can be performed on the basis of additional knowledge of so-called *causal relationships*. These define how the occurrence of a particular atomic effect might cause the additional occurrence of certain other effects. The intention is to apply such rules if and as long as the situation at hand (which has been obtained by computing the direct effects of an action) violates the domain constraints. Second, we illustrate how causal relationships can be automatically generated from given domain constraints by taking into account general knowledge of how facts can possibly affect each other.

The applicability of our solution is demonstrated by extending a concrete STRIPS-like yet purely deductive method which is based on Equational Logic Programming (ELP) [Hölldobler and Schneeberger, 1990]. In contrast to STRIPS, this approach has recently turned out to have a wide range of applicability as regards general aspects of topical interest, e.g., postdiction problems, reasoning about nondeterministic and concurrent actions, etc. [Thielscher, 1994; Bornscheuer and Thielscher, 1994]. Moreover, this method is provably equivalent to a modification of the Connection Method designed for planning problems [Bibel, 1986] and to an approach to planning based on Linear Logic [Masseron *et al.*, 1990] (see [Große *et al.*, 1995]).

The rest of this paper is organized as follows. In Section 2, we introduce the formal notion of a causal relationship. Based on this concept, we define post-processing steps used to compute ramifications according to underlying domain constraints. In Section 3, we present an algorithm to generate causal relationships automatically given some domain constraints and a formal specification of which fluents are causally connected. In Section 4, we turn to the ELP-based approach and show how to integrate our solution to the ramification problem. To this end, we extend the basic logic program by clauses expressing domain constraints and modeling the successive application of single causal relationships. The adequateness of this extension is proved

wrt the semantics developed in Section 2. Finally, our proposal is discussed and future research is outlined in Section 5. In particular, we compare the employment of causal relationships with methods based on the distinction between frame and non-frame fluents made to address the ramification problem [Lifschitz, 1990; Kartha and Lifschitz, 1994]—and show the limitedness of the latter.

2 Applying Causal Relationships

A formal framework to reason about actions and change requires the basic notion of a *situation*, which is a partial snapshot of the world at a particular instant of time. So-called *fluents* serve as the atomic elements to describe situations—these are properties whose truth values may change in the course of time [McCarthy and Hayes, 1969]. For sake of simplicity, we assume a finite number of propositional constants as the underlying set of fluents throughout this paper.

Definition 1 Let $\mathcal{F}$ be a finite set of symbols, called *fluents*. A *situation* $\mathcal{S}$ is a set of *fluent literals*, i.e., expressions of the form f or $\overline{f}$, where $f \in \mathcal{F}$, such that each fluent occurs either affirmatively (f) or negatively ($\overline{f}$) in $\mathcal{S}$. ■

For convenience, let us agree to interpret $\overline{\overline{f}}$ as f, and by $|l|$ we denote the affirmative part of a fluent literal l, i.e., $|f| = |\overline{f}| = f$ where $f \in \mathcal{F}$.

Example 1 The basic version of the Yale Shooting scenario [Hanks and McDermott, 1987] consists of the fluents *loaded* and *alive*, which determine the state of a gun and a turkey, respectively. For instance, the situation $\{\overline{loaded}, alive\}$ describes the fact that the gun is unloaded and the turkey is alive. ■

Example 2 The Electric Circuit domain [Lifschitz, 1990] consists of two binary switches whose positions are described by the fluents $switch_1$ and $switch_2$, respectively, plus a light bulb represented by the fluent *light*. The set $\{\overline{switch_1}, switch_2, \overline{light}\}$, for instance, describes a situation where the two switches are in different positions and the light is off. ■

Actions are specified in a STRIPS-like fashion by stating a set of fluent literals to be removed from along with a set of fluent literals to be added to the situation at hand. Such an action is applicable if all literals to be removed are contained in the current situation:

Definition 2 Let $\mathcal{F}$ be a set of fluents. An *action description* is a triple $\langle \mathcal{C}, a, \mathcal{E} \rangle$ where $\mathcal{C}$, called *condition*, and $\mathcal{E}$, called *effect*, are sets of fluent literals and the *action name* a is a symbol. Such an action description is *applicable* in a situation $\mathcal{S}$ iff $\mathcal{C} \subseteq \mathcal{S}$. The application yields the set $(\mathcal{S} \setminus \mathcal{C}) \cup \mathcal{E}$. ■

Example 1 (continued) The action of loading the gun can be formalized by the triple

$$\langle \{\overline{loaded}\}, load, \{loaded\} \rangle . \tag{1}$$

This action description is applicable in the situation $\mathcal{S} = \{\overline{loaded}, alive\}$ since $\{\overline{loaded}\} \subseteq \mathcal{S}$; its application yields $(\mathcal{S} \setminus \{\overline{loaded}\}) \cup \{loaded\} = \{loaded, alive\}$. ■

Since the effects of an action may vary from situation to situation, one often needs more than just a single specification such as (1). In order to avoid unintended conflicts in case of multiple descriptions of one action, we employ the following partial order on action descriptions [Hölldobler and Thielscher, 1995]:

Definition 3 An action description $\langle \mathcal{C}_1, a_1, \mathcal{E}_1 \rangle$ is *more specific* than a description $\langle \mathcal{C}_2, a_2, \mathcal{E}_2 \rangle$ iff $a_1 = a_2$ and $\mathcal{C}_1 \supset \mathcal{C}_2$. If $\mathcal{A}$ is a set of action descriptions, $\mathcal{S}$ a situation and a an action name then a *successor* of $\mathcal{S}$ is obtained by applying the most specific (wrt $\mathcal{A}$) applicable action description of a to $\mathcal{S}$. ■

Example 1 (continued) The action of shooting with the gun might be defined by

$$\langle \{loaded\}, shoot, \{\overline{loaded}\} \rangle \tag{2}$$

stating that the gun becomes unloaded. Additionally, shooting at a living turkey with a previously loaded gun causes him to drop dead, i.e.,

$$\langle \{loaded, alive\}, shoot, \{\overline{loaded}, \overline{alive}\} \rangle . \tag{3}$$

Following Definition 2, both (2) and (3) are applicable in the situation $\{loaded, alive\}$. However, (3) is more specific than (2) and, hence, the only successor of $\mathcal{S}$ is $\{\overline{loaded}, \overline{alive}\}$ as intended. ■

In the sequel, we assume that any given set of action descriptions is deterministic, i.e., contains at most one most specific applicable description for each action and situation. Furthermore, its application shall always preserve the condition of being a situation (see Definition 1).

Our concept of a situation suggests that first of all each possible combination of fluent values describes a possible situation in the domain being modeled. In more complex scenarios, however, fluents might depend on each other. These dependencies are expressed by means of *domain constraints*, which are formalized as (propositional) formulae built up from fluent literals and the usual logical connectives:

Definition 4 Let $\mathcal{F}$ be a set of fluents. The set of *fluent formulae* is the smallest set such that the constant $\top$ (*true*) and each fluent literal f and $\overline{f}$ ($f \in \mathcal{F}$) are fluent formulae; and if F and G are fluent formulae then $(F \wedge G)$, $(F \vee G)$, $(F \Rightarrow G)$, and $(F \Leftrightarrow G)$ are also fluent formulae.

If $\mathcal{S}$ is a situation and F a fluent formula then the notion of F being *true* in $\mathcal{S}$, written $\mathcal{S} \models F$, is inductively defined as follows:

1. $\mathcal{S} \models \top$;
2. $\mathcal{S} \models f$ (resp. $\mathcal{S} \models \overline{f}$) iff $f \in \mathcal{S}$ (resp. $\overline{f} \in \mathcal{S}$), for each $f \in \mathcal{F}$;
3. $\mathcal{S} \models (F \wedge G)$ iff both $\mathcal{S} \models F$ and $\mathcal{S} \models G$;
4. $\mathcal{S} \models (F \vee G)$ iff $\mathcal{S} \models F$ or $\mathcal{S} \models G$ (or both);
5. $\mathcal{S} \models (F \Rightarrow G)$ iff $\mathcal{S} \not\models F$ or $\mathcal{S} \models G$ (or both); and
6. $\mathcal{S} \models (F \Leftrightarrow G)$ iff $\mathcal{S} \models F$ and $\mathcal{S} \models G$ or else $\mathcal{S} \not\models F$ and $\mathcal{S} \not\models G$.

■

Example 1 (continued) The Yale Shooting scenario is extended by two fluents, *dead* and *walking*, stating whether the turkey is dead and whether he is walking around, respectively [Baker, 1991]. The constraint $alive \Leftrightarrow \overline{dead}$ then formalizes the obvious connection between these two fluents while $walking \Rightarrow alive$ states that our turkey must be alive to take some exercises. ■

Example 2 (continued) We assume that our electric circuit is designed such that the light bulb is on if and only if the two switches are in the same position (see also Figure 1 in Section 5). This is expressed via the domain constraint $light \Leftrightarrow (switch_1 \Leftrightarrow switch_2)$. ■

The ramification problem now arises as soon as domain constraints describe dependencies which are not reflected in some action description. Since the frame assumption tells us that fluents keep their value unless they are explicitly mentioned in such an action description, the domain constraints become violated through the execution of that action. For instance, applying (3) to the (reasonable) situation $\{loaded, alive, \overline{dead}, walking\}$ yields the set $\mathcal{S}' = \{\overline{loaded}, \overline{alive}, \overline{dead}, walking\}$ according to Definition 2, which now violates both underlying domain constraints from above.

From a theoretical point of view it is of course conceivable to integrate all domain constraints into the various action descriptions, by making intensive use of our specificity criterion. From a practical point of view, however, this would soon lead to an intractable number of action descriptions when trying to model slightly more complex scenarios. Moreover, adding a new domain constraint to a scenario might force, in the worst case, the reconstruction of a completely new set of action descriptions.

Instead of exhaustively integrating domain constraints into action descriptions, we should view these constraints as reason for certain indirect effects in addition to the effects explicitly occurring when executing actions according to Definition 2. These indirect effects are computed by performing additional post-processing steps. Hence, the basic idea is to regard a set like $\mathcal{S}' = \{\overline{loaded}, \overline{alive}, \overline{dead}, walking\}$ from above as a mere approximation which requires further investigation to obtain the finally resulting situation. If and as long as the preliminarily obtained set violates the given domain constraints, particular indirect effects are generated. To this end, we need information of the various causal connections between the fluents in the domain under consideration. For instance, our domain constraint $alive \Leftrightarrow \overline{dead}$ suggests that if some action affects the fluent *alive* then it also influences, indirectly, the fluent *dead*. In particular, this information should lead to the replacement of $\overline{dead}$ by *dead* in $\mathcal{S}'$. Such a replacement—triggered by what will be called a *causal relationship* below—is considered as a single step in the post-process to be performed in order to address ramifications. It is of course inevitably necessary to know which of the two fluents $\overline{alive}$ and $\overline{dead}$ in $\mathcal{S}'$ was an immediate effect of the action that has just been executed. Otherwise, one could as well argue that the occurrence of $\overline{dead}$ should lead to the replacement of $\overline{alive}$ by *alive*, which is clearly not intended in this example. Moreover, since indirect effects potentially cause additional effects, it is necessary to keep in mind all computed effects during the post-process. The following definition of a causal relationship and its application to a set of fluents in conjunction with a particular set of effects makes this idea manifest:

Definition 5 A *causal relationship* is an expression of the form $C : e \rightsquigarrow f$ where C is a fluent formula and e, f are fluent literals. Let $\mathcal{S}$ be a situation and $\mathcal{E}$ a set of fluent literals then such a relationship is *applicable* to $(\mathcal{S}, \mathcal{E})$ iff $\mathcal{S} \models C$, $\overline{f} \in \mathcal{S}$ and $e \in \mathcal{E}$. Its application yields $\mathcal{S}' = (\mathcal{S} \setminus \{\overline{f}\}) \cup \{f\}$ and $\mathcal{E}' = \mathcal{E} \cup \{f\}$.

Let $\mathcal{S}$ be a consistent (wrt a set of given domain constraints $\mathcal{D}$) situation and $\langle \mathcal{C}, a, \mathcal{E} \rangle$ be a most specific applicable action description (wrt a fixed set $\mathcal{A}$). A consistent (wrt $\mathcal{D}$) set $\mathcal{S}'$ is a *result* of executing a in $\mathcal{S}$ if some $(\mathcal{S}', \mathcal{E}')$ can be obtained by applying a sequence of causal relationships to $((\mathcal{S} \setminus \mathcal{C}) \cup \mathcal{E}, \mathcal{E})$. ■

In words, a causal relationship consists of a condition C that has to be satisfied, a particular effect e that must have been occurred (directly or indirectly), and a fluent that changes its value as an indirect effect caused by e. A resulting situation is then obtained by applying a pure action description following Definition 3 and, afterwards, using causal relationships until the underlying domain constraints become satisfied.

Example 1 (continued) Let the following two causal relationships be given:

$$\begin{array}{lccc} \top: & \overline{alive} & \rightsquigarrow & dead \\ \top: & \overline{alive} & \rightsquigarrow & \overline{walking} \end{array} \tag{4}$$

If $\mathcal{S} = \{loaded, alive, \overline{dead}, walking\}$ denotes the situation at hand then the application of (3) yields the set $(\mathcal{S} \setminus \mathcal{C}) \cup \mathcal{E} = \{\overline{loaded}, \overline{alive}, \overline{dead}, walking\}$ along with $\mathcal{E} = \{\overline{loaded}, \overline{alive}\}$. Now, the two relationships (4) can be sequentially applied as follows:

$$\begin{array}{c} (\,\{\overline{loaded}, \overline{alive}, \overline{dead}, walking\}\,,\, \mathcal{E}\,) \\ \downarrow \\ (\,\{\overline{loaded}, \overline{alive}, dead, walking\}\,,\, \mathcal{E} \cup \{dead\}\,) \\ \downarrow \\ (\,\{\overline{loaded}, \overline{alive}, dead, \overline{walking}\}\,,\, \mathcal{E} \cup \{dead, \overline{walking}\}\,) \end{array}$$

The resulting situation does no longer violate the domain constraints and is exactly the intended solution. ■

Example 2 (continued) Consider the relationship

$$switch_2 : \; switch_1 \rightsquigarrow light\,. \tag{5}$$

Furthermore, let $\langle \overline{switch_1}, toggle_1, switch_1 \rangle$ be one specification of toggling the first switch. This description is applicable in $\{\overline{switch_1}, switch_2, \overline{light}\}$ and yields $\mathcal{S}' = \{switch_1, switch_2, \overline{light}\}$ along with the set of effects $\mathcal{E} = \{switch_1\}$. Because of $\mathcal{S}' \models switch_2$, the causal relationship (5) is applicable and yields $\{switch_1, switch_2, light\}$, which now is compatible with the underlying domain constraint. ■

The reader should be aware of the fact that Definition 5 does neither guarantee uniqueness nor even existence of a successor situation. Both aspects will be discussed in Section 5.

3 Extracting Causal Relationships

The causal relationships used in the previous section were given as part of the problem specification. In this section, we investigate the possibility to extract these rules automatically from arbitrary domain constraints. Doing this, however, one is faced with a well-known problem: Recall the formula $light \Leftrightarrow (switch_1 \Leftrightarrow switch_2)$, and consider the set of fluents $\{switch_1, switch_2, \overline{light}\}$ being obtained by changing the position of the first switch (Example 2). This situation violates the underlying constraint. To correct this via a ramifications step, there are two different possibilities even in case we are aware of the fact that the first switch has just changed its position. Either we could replace $\overline{light}$ by $light$ or else $switch_2$ by $\overline{switch_2}$. Both are (minimal) changes resulting in a situation that satisfies our domain constraint. Obviously, the latter contradicts our intuition; yet no syntactical criterion tells us which of the two changes should be preferred. By using the given, intended causal relationship (5), we have obtained the desired result. But first of all our constraint suggests as well the counterintuitive relationship $\overline{light} : switch_1 \rightsquigarrow \overline{switch_2}$.

This is a general problem arising in any formal approach to the ramification problem. Several ways to overcome it are informally discussed in [Ginsberg and Smith, 1988]. The most promising and reasonable solution, which we adapt here, seems to be the incorporation of additional knowledge of the domain into the reasoning process. For instance, we know from general laws of physics that changing a switch's position might immediately influence the bulb but not another switch.[1]

Domain knowledge of how fluents possibly affect each other is formally specified by a binary *influence* relation $\mathcal{I}$ on the set of fluents. For example, defining $\mathcal{I} = \{(switch_1, light), (switch_2, light)\}$ encodes the fact that a change of a switch's position can possibly influence the state of the light bulb but not vice versa. Based on this additional information, all intended causal relationships can be extracted automatically from given domain constraints by considering each possible way to satisfy a violated constraint through the change of certain fluents:

Algorithm 1 Let $\mathcal{F}$ be a set of fluents, $\mathcal{D}$ a set of domain constraints and $\mathcal{I} \subseteq \mathcal{F} \times \mathcal{F}$ an influence relation. For each $D \in \mathcal{D}$ do:

1. Let $F_1 \wedge \ldots \wedge F_n$ be the conjunctive normal form (CNF) of D ($n \geq 1$). For each $i = 1, \ldots, n$ do:
2. Let $F_i = L_1 \vee \ldots \vee L_m$ where each L_j is a fluent literal ($m \geq 1$). For each $j = 1, \ldots, m$ do:
3. For each $k = 1, \ldots, m$ such that $j \neq k$ and $(|L_j|, |L_k|) \in \mathcal{I}$ generate this causal relationship:

$$\bigwedge_{\substack{l=1,\ldots,m \\ l \neq j,\, l \neq k}} \overline{L_l} \;:\; \overline{L_j} \rightsquigarrow L_k\,.$$

■

[1] It is important to distinguish between immediate and possibly indirect influence. Though two switches cannot directly affect each other, they might be causally connected indirectly through, for instance, a relay (see also Figure 1).

Note that a domain constraint might yield an exponential number of causal relationships if its CNF consists of exponentially many literals. But the overall number of relationships is linear in the size of $\mathcal{D}$ because each constraint is treated separately.

Example 2 (continued) Consider the domain constraint $D = light \Leftrightarrow (switch_1 \Leftrightarrow switch_2)$ and the influence relation $\mathcal{I} = \{(switch_1, light), (switch_2, light)\}$. The CNF of D is

$$(light \vee \overline{switch_1} \vee \overline{switch_2}) \wedge (light \vee switch_1 \vee switch_2) \\ \wedge (\overline{light} \vee \overline{switch_1} \vee switch_2) \wedge (\overline{light} \vee switch_1 \vee \overline{switch_2})\,.$$

Let us apply the second and third step of Algorithm 1 to the first conjunct, $light \vee \overline{switch_1} \vee \overline{switch_2}$:

- In case $L_{j=1} = light$, no causal relationship is generated since $\mathcal{I}$ includes neither $(light, switch_1)$ nor $(light, switch_2)$.
- In case $L_{j=2} = \overline{switch_1}$, this causal relationship is generated due to $(switch_1, |L_{k=1}| = light) \in \mathcal{I}$:

  $$switch_2 : \; switch_1 \rightsquigarrow light$$

  (c.f. (5)), whereas $(switch_1, |L_{k=3}| = switch_2) \notin \mathcal{I}$ prevents us from generating the unintended relationship $\overline{light} : switch_1 \rightsquigarrow \overline{switch_2}$.
- In case $L_{j=3} = \overline{switch_2}$, this causal relationship is generated due to $(switch_2, |L_{k=1}| = light) \in \mathcal{I}$:

  $$switch_1 : \; switch_2 \rightsquigarrow light$$

  while $(switch_2, |L_{k=2}| = switch_1) \notin \mathcal{I}$ suppresses the generation of another one.

Correspondingly, the remaining conjuncts in the CNF above yield these six causal relationships:

$$\begin{array}{llll} \overline{switch_2} : & \overline{switch_1} \rightsquigarrow light & \quad \overline{switch_1} : & \overline{switch_2} \rightsquigarrow light \\ \overline{switch_2} : & switch_1 \rightsquigarrow \overline{light} & \quad switch_1 : & \overline{switch_2} \rightsquigarrow \overline{light} \\ switch_2 : & \overline{switch_1} \rightsquigarrow \overline{light} & \quad \overline{switch_1} : & switch_2 \rightsquigarrow \overline{light} \end{array}$$

■

Example 1 (continued) Given the influence relation $\mathcal{I} = \{(alive, dead), (dead, alive), (alive, walking)\}$, the reader is invited to verify that our two domain constraints yield these five causal relationships:

$$\begin{array}{llll} \top : & alive \rightsquigarrow \overline{dead} & \quad \top : & \overline{alive} \rightsquigarrow dead \\ \top : & dead \rightsquigarrow \overline{alive} & \quad \top : & \overline{dead} \rightsquigarrow alive \\ \top : & \overline{alive} \rightsquigarrow \overline{walking} & & \end{array} \qquad (6)$$

■

4 Ramification in ELP

The completely reified representation of situations is the distinguishing feature of the ELP-based approach [Hölldobler and Schneeberger, 1990]. To this end, the fluent literals being true in a situation are treated as terms and are connected using a special binary function symbol, denoted by $\circ$ and written in infix notation:

Definition 6 Let $\mathcal{S} = \{f_1, \ldots, f_m, \overline{f_{m+1}}, \ldots, \overline{f_n}\}$ be a set of fluent literals then

$$\tau_{\mathcal{S}} \stackrel{\text{def}}{=} f_1 \circ \ldots \circ f_m \circ \overline{f_{m+1}} \circ \ldots \circ \overline{f_n}$$

is the *term representation* of $\mathcal{S}$. ■

Intuitively, the order of the fluent literals occurring in a term representation should be irrelevant, which is why we employ an underlying equational theory, viz

$$\begin{aligned} (X \circ Y) \circ Z &=_{\mathrm{AC1}} X \circ (Y \circ Z) && \text{(A)} \\ X \circ Y &=_{\mathrm{AC1}} Y \circ X && \text{(C)} \\ X \circ \emptyset &=_{\mathrm{AC1}} X && (1) \end{aligned}$$

where $\emptyset$ is a constant denoting a unit element for $\circ$.

Based on this equational theory, written (AC1), the reasoning process described in Definitions 2 and 3 is encoded by means of an equational logic program (see, e.g., [Hölldobler, 1989]) as follows. Each action description $\langle \mathcal{C}, a, \mathcal{E} \rangle$ is represented by the unit clause $\mathit{action}(\tau_{\mathcal{C}}, a, \tau_{\mathcal{E}})$. The application of actions, which includes consideration of the specificity criterion, is performed on the basis of these three program clauses:[2]

$$\begin{aligned} &\mathit{causes}(I, [\,], I). \\ &\mathit{causes}(I, [A|P], G) \leftarrow \mathit{action}(C, A, E), \\ &\qquad C \circ V =_{\mathrm{AC1}} I, \\ &\qquad \neg \mathit{non\_specific}(A, C, I), \\ &\qquad \mathit{causes}(V \circ E, P, G). \qquad (7) \\ &\mathit{non\_specific}(A, C, I) \leftarrow \mathit{action}(C', A, E'), \\ &\qquad C' \circ V =_{\mathrm{AC1}} I, \\ &\qquad C \circ W =_{\mathrm{AC1}} C', \\ &\qquad \neg W =_{\mathrm{AC1}} \emptyset. \end{aligned}$$

In words, the ternary predicate $\mathit{causes}(i, [a_1, \ldots, a_n], g)$ expresses the fact that the application of the sequence $[a_1, \ldots, a_n]$ of actions to the initial situation i yields the goal situation g. If $n \geq 1$ then an action description of a_1 with condition c and effect e is selected such that c is contained in i, i.e., $\exists V. c \circ V =_{\mathrm{AC1}} i$; the description is most specific wrt i; and the recursive call uses the resulting situation $V \circ e$ along with the sequence $p = [a_2, \ldots, a_n]$. An instance $\mathit{non\_specific}(a, c, i)$ states the existence of a description for a which is more specific wrt the situation i than the particular description with condition c. Following Definition 3, such a more specific action description must have some condition c' such that c' is contained in i and the set corresponding to c' is a strict superset of the set corresponding to c.[3]

The clauses in (7) plus the ones encoding a given set of action descriptions in conjunction with the equational theory (AC1) form the basic ELP to reasoning about actions. Since this logic program does not only require a special unification procedure but also contains negative literals, an adequate computation mechanism is *SLDENF-resolution*, i.e., SLD-resolution augmented by theory unification and the negation-as-failure principle to handle negative subgoals [Shepherdson, 1992; Thielscher, 1995]. In [Hölldobler and Thielscher, 1995], the application of this calculus to our ELP has been proved adequate wrt the semantics given by Definitions 1–3.[4]

[2] We use a PROLOG-like syntax, i.e., constants and predicates are in lower cases whereas variables are denoted by upper case letters. As usual, the term $[h \,|\, t]$ denotes a list with head h and tail t.

[3] In order to treat equality subgoals, we implicitly add the clause $X =_{\mathrm{AC1}} X$ encoding *reflexivity*.

[4] It is important to realize that the axioms (AC1) essentially model the data structure multiset rather than sets; the necessity of this sophistication, as regards our program (7), has been shown in [Große *et al.*, 1995]. We neglect this difference here—which is justified by results presented in [Thielscher, 1994].

4.1 Integrating Domain Constraints

As a first step towards integrating our solution to the ramification problem into the ELP-based approach, we have to provide a way to express domain constraints. To this end, we overload to a certain extent the four logical connectives $\wedge$, $\vee$, $\Rightarrow$ and $\Leftrightarrow$, introduced in Definition 4, by treating them as special binary functions in our logic program. Given a fluent formula F, by τ_F we denote its term representation based on these functions (and where the special formula $\top$ is represented by the unit element $\emptyset$). The following clauses encode the notion of a formula being true in a particular situation:

$$\begin{aligned} &\mathit{holds}(F, F \circ V). \\ &\mathit{holds}(F \wedge G, S) \leftarrow \mathit{holds}(F, S), \mathit{holds}(G, S). \\ &\mathit{holds}(F \vee G, S) \leftarrow \mathit{holds}(F, S). \\ &\mathit{holds}(F \vee G, S) \leftarrow \mathit{holds}(G, S). \qquad (8) \\ &\mathit{holds}(F \Rightarrow G, S) \leftarrow \neg \mathit{holds}(F, S). \\ &\mathit{holds}(F \Rightarrow G, S) \leftarrow \mathit{holds}(G, S). \\ &\mathit{holds}(F \Leftrightarrow G, S) \leftarrow \mathit{holds}(F \Rightarrow G, S), \mathit{holds}(G \Rightarrow F, S). \end{aligned}$$

These clauses are adequate wrt Definition 4:

Proposition 7 *Let $\mathcal{F}$ be a set of fluents, F a fluent formula and $\mathcal{S}$ a situation. Furthermore, let $\tau_{\mathcal{S}}$ and τ_F denote term representations of $\mathcal{S}$ and F, respectively, then $\mathcal{S} \models F$ iff (8) $\cup \{ \leftarrow \mathit{holds}(\tau_F, \tau_{\mathcal{S}}) \}$ has an SLDENF-refutation wrt (AC1).*

Proof: If $F = \top$ then $\mathcal{S} \models F$. Correspondingly, the query $\leftarrow \mathit{holds}(\emptyset, \tau_{\mathcal{S}})$ can be refuted by applying the first clause in (8) since $\tau_{\mathcal{S}} =_{\mathrm{AC1}} \emptyset \circ \tau_{\mathcal{S}}$.

If $F = f$ (resp. $F = \overline{f}$) for some $f \in \mathcal{F}$ then $\mathcal{S} \models F$ iff $f \in \mathcal{S}$ (resp. $\overline{f} \in \mathcal{S}$). Correspondingly, the query $\leftarrow \mathit{holds}(\tau_F, \tau_{\mathcal{S}})$ can only be resolved by applying the first clause in (8) and only if $\tau_{\mathcal{S}}$ and $f \circ V$ (resp. $\overline{f} \circ V$) are (AC1)-unifiable.

The various cases where F is a complex formula can be easily proved by induction on its structure.[5] ∎

Based on this formalization, an underlying set of domain constraints $\{D_1, \ldots, D_n\}$ is used to define consistency of situations via the following program clause:

$$\mathit{consistent}(S) \leftarrow \mathit{holds}(\tau_{D_1}, S), \ldots, \mathit{holds}(\tau_{D_n}, S). \qquad (9)$$

4.2 Causal Relationships in ELP

To integrate into the ELP-based approach causal relationships and their application in order to address the ramification problem, we introduce a new predicate $\mathit{ramify}(\tau_{\mathcal{S}}, \tau_{\mathcal{E}}, \tau_{\mathcal{S}'}, \tau_{\mathcal{E}'})$. Its intended meaning is that the application of some causal relationship to a pair $(\mathcal{S}, \mathcal{E})$ yields a pair $(\mathcal{S}', \mathcal{E}')$. More precisely, each causal relationship $C : e \rightsquigarrow f$ is encoded via the unit clause

$$\mathit{ramify}(S \circ \overline{f}, E \circ e, S \circ f, E \circ e \circ f) \leftarrow \mathit{holds}(\tau_C, S \circ \overline{f}). \qquad (10)$$

In words, if the situation at hand contains the fluent literal $\overline{f}$ and entails C, and if in addition e has occurred

[5] Regarding the fifth clause in (8), note that a subgoal $\neg \mathit{holds}(\tau_F, \tau_{\mathcal{S}})$ has an SLDENF-refutation iff the affirmative query $\leftarrow \mathit{holds}(\tau_F, \tau_{\mathcal{S}})$ finitely fails, i.e., is not refutable.

as effect then f is replaced by f'. Furthermore, f' is added to the current set of effects. For brevity, given a set $\mathcal{R}$ of causal relationships by $(10)_{\mathcal{R}}$ we denote the set of program clauses encoding the elements of $\mathcal{R}$.

In order to apply causal relationships in a post-process after having applied an action description, the body of the second clause defining *causes* in our original program, (7), is extended by a literal named *consistify*:

$$\begin{array}{lll} \mathit{causes}(I,[A|P],G) & \leftarrow & \mathit{action}(C,A,E),\\ & & C \circ V =_{\mathrm{AC1}} I,\\ & & \neg \mathit{non\_specific}(A,C,I),\\ & & \mathit{consistify}(V \circ E, E, S),\\ & & \mathit{causes}(S,P,G). \end{array} \qquad (11)$$

An instance $\mathit{consistify}(v \circ e, e, s)$ is intended to express the fact that the preliminarily computed (possibly inconsistent) situation $v \circ e$ along with the effect e can be transformed into a resulting (consistent) situation s through the successive application of causal relationships (c.f. Definition 5):

$$\begin{array}{lll} \mathit{consistify}(S,E,S) & \leftarrow & \mathit{consistent}(S).\\ \mathit{consistify}(S,E,T) & \leftarrow & \mathit{ramify}(S,E,S',E'),\\ & & \mathit{consistify}(S',E',T). \end{array} \qquad (12)$$

In words, if the situation at hand is already consistent (c.f. (9)) then it describes an acceptable result. On the other hand, we can apply a causal relationship to the situation S along with the effects E that yields a new situation S' and a new set of effects E' according to Definition 5, which are then used in a recursive call.

The following main result states the adequateness of our extended program wrt the proposal of Section 2:

Theorem 8 *Let $\mathcal{F}$ be a set of fluents, $\mathcal{R}$ a set of causal relationships, $\mathcal{D}$ a set of domain constraints and $\mathcal{S}, \mathcal{E}, \mathcal{S}'$ three sets of fluents. If $\mathcal{S}$ is the outcome of having applied some most specific action description with effect $\mathcal{E}$ to some consistent situation then $\mathcal{S}'$ is a resulting situation iff $\leftarrow \mathit{consistify}(\tau_{\mathcal{S}}, \tau_{\mathcal{E}}, \tau_{\mathcal{S}'})$ has an SLDENF-refutation wrt $(9)_{\mathcal{D}} \cup (10)_{\mathcal{R}} \cup (12)$ and (AC1).*

Proof (sketch): There is a one-to-one correspondence between the application of a causal relationship according to Definition 5 and the application of the corresponding clause (10). Furthermore, the first clause in (12) in conjunction with Proposition 7 requires $\mathcal{S}'$ to satisfy $\mathcal{D}$. The formal proof is by induction on the number of applied causal relationships and the length of the refutation, respectively. ■

Example 1 (continued) Given the encodings of the two relationships in (4), of the underlying domain constraints, and of the action description (3), i.e.,

$$\mathit{ramify}(S \circ \overline{\mathit{dead}}, E \circ \overline{\mathit{alive}}, S \circ \mathit{dead}, E \circ \overline{\mathit{alive}} \circ \mathit{dead}).$$

$$\begin{array}{l} \mathit{ramify}(S \circ \mathit{walking}, E \circ \overline{\mathit{alive}},\\ \qquad\qquad S \circ \overline{\mathit{walking}}, E \circ \overline{\mathit{alive}} \circ \overline{\mathit{walking}}). \end{array}$$

$$\begin{array}{lll} \mathit{consistent}(S) & \leftarrow & \mathit{holds}(\mathit{alive} \Leftrightarrow \overline{\mathit{dead}}, S),\\ & & \mathit{holds}(\mathit{walking} \Rightarrow \mathit{alive}, S). \end{array}$$

$$\mathit{action}(\mathit{loaded} \circ \mathit{alive}, \mathit{shoot}, \overline{\mathit{loaded}} \circ \overline{\mathit{alive}}).$$

it is possible to find an SLDENF-refutation for the query $\leftarrow \mathit{causes}(\mathit{loaded} \circ \mathit{alive} \circ \overline{\mathit{dead}} \circ \mathit{walking}, [\mathit{shoot}], G)$ with answer $G \mapsto \overline{\mathit{loaded}} \circ \overline{\mathit{alive}} \circ \mathit{dead} \circ \overline{\mathit{walking}}$. ■

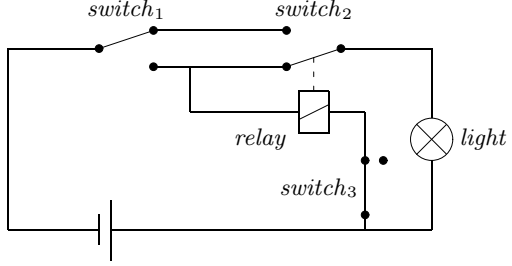

Figure 1: An extended electric circuit consisting of five fluents. The current state is described by $\overline{\mathit{switch}_1}$ (the first switch is up), switch_2 (the second switch is down), switch_3 (the third switch is closed), $\overline{\mathit{light}}$ (the light bulb is off) and $\overline{\mathit{relay}}$ (the relay is deactivated).

5 Discussion

We have proposed the application of single causal relationships in a post-processing step in order to address the ramification problem. Moreover, it has been illustrated how an adequate set of such relationships can be automatically extracted from given domain constraints by taking into account general knowledge of how fluents can possibly affect each other. Finally, we have integrated both domain constraints and our method to comply with them into an approach to reason about actions and change based on Equational Logic Programming.

The merits of causal relationships shall be illustrated by an example that shows the limitedness of a common, simpler way to avoid unintended ramifications, namely, the distinction between so-called *frame* and *non-frame* fluents [Lifschitz, 1990; Kartha and Lifschitz, 1994]:[6] Consider the extended Electric Circuit scenario depicted in Figure 1. Two switches and the light bulb are related as usual, i.e., $\mathit{light} \Leftrightarrow (\mathit{switch}_1 \Leftrightarrow \mathit{switch}_2)$; the additional relay is controlled by the first and third switch through $\mathit{relay} \Leftrightarrow \mathit{switch}_1 \wedge \mathit{switch}_3$; and the relay is intended to force the second switch into the upper position if activated, i.e., $\mathit{relay} \Rightarrow \overline{\mathit{switch}_2}$. Now, in order to prefer the light bulb changing its state to a switch unexpectedly jumping its position (see Section 3), we are supposed to consider switch_1 and switch_2 frame and *light* non-frame. Accordingly, switch_3 and *relay* should be considered frame and non-frame, respectively. However, if we take the situation depicted in Figure 1 and change the position of the first switch then it is impossible to distinguish between these two outcomes: Either the relay becomes activated and causes the second switch to jump into the upper position (as intended) or else the relay remains deactivated and the third switch opens magically! This is because in both cases a second frame fluent (aside from switch_1) is affected, yet no preference is discernible. In contrast, and the reader is invited to verify this, given the natural influence relation $\mathcal{I} = \{(\mathit{switch}_1, \mathit{light}), (\mathit{switch}_2, \mathit{light}), (\mathit{switch}_1, \mathit{relay}),$

[6] Roughly spoken, the idea there is to prefer changes of non-frame fluents to changes of frame fluents whenever additional effects have to be considered in order to satisfy domain constraints.

$(switch_3, relay)$, $(relay, switch_2)\}$, Algorithm 1 generates an adequate set of causal relationships whose application according to Definition 5 yields the intended, unique resulting situation.

A second merit of causal relationships is that they enable us to comply with the observation that domain constraints might give rise to *qualifications* rather than ramifications [Lin and Reiter, 1994]. Consider, for instance, the action of enticing the turkey to walk.[7] The effect of executing this action shall be $\{walking\}$. Yet in case the turkey is not alive, the constraint $walking \Rightarrow alive$ should clearly not lead to his revival. Rather it imposes an additional qualification on our new action. This is reflected in the causal relationships (6): None of them is applicable to $\mathcal{S} = \{\overline{alive}, dead, walking\}$, $\mathcal{E} = \{walking\}$; hence, no resulting situation is obtained here following Definition 5. Our logic program developed in Section 4 behaves accordingly—the reader is invited to verify that $\leftarrow causes(\overline{alive} \circ dead \circ \overline{walking}, [entice], G)$ has no SLDENF-refutation given the encoding of (6) and the action description $action(\overline{walking}, entice, walking)$.

The example just discussed illustrates the possibility that a resulting situation does not at all exist, i.e., the available causal relationships might not always be sufficient to satisfy the underlying domain constraints. As we have seen, this induces (additional) qualifications. On the other hand, it is also conceivable that multiple successor situations can be obtained by applying different sequences of causal relationships all resulting in the satisfaction of the domain constraints. This induces nondeterminism. Both non-existence as well as multitude of resulting situations rise the important question of general criteria that guarantee a unique successor. A comparison with the concept of *revision programs*, introduced in [Marek and Truszczyński, 1994], might help to address this problem. Revision programs along with their application in order to modify states of databases resemble causal relationships and their application in order to compute ramifications. In [Marek and Truszczyński, 1995], two syntactic conditions have been proved guarantee of a unique revised database. Whether and how these results can be adapted to our approach depends on the formal correspondence between the two concepts and is left as future work.

Acknowledgments

The author is grateful to Christoph Herrmann and the anonymous referees for helpful comments on an earlier version of this paper. Thanks also to the participants of the AAAI 95 Spring Symposium on Theories of Actions for several interesting discussions related to the work reported here.

References

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

[Bibel, 1986] W. Bibel. A Deductive Solution for Plan Generation. *New Generation Computing*, 4:115–132, 1986.

[Bornscheuer and Thielscher, 1994] S.-E. Bornscheuer and M. Thielscher. Representing Concurrent Actions and Solving Conflicts. In B. Nebel and L. Dreschler-Fischer, ed.'s, *KI-94: Advances in Artif. Intell.*, vol. 861 of *LNAI*, p. 16–27, Saarbrücken, Germany, Sep. 1994. Springer.

[Fikes and Nilsson, 1971] R. Fikes and N. Nilsson. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Artif. Intell.*, 2:189–208, 1971.

[Finger, 1987] J. J. Finger. *Exploiting constraints in design synthesis.* PhD thesis, Stanford University, 1987.

[Ginsberg and Smith, 1988] M. L. Ginsberg and D. E. Smith. Reasoning about Action I: A possible worlds approach. *Artif. Intell.*, 35:165–195, 1988.

[Große *et al.*, 1995] G. Große, S. Hölldobler, and J. Schneeberger. Linear Deductive Planning. *J. of Logic and Computation*, 1995.

[Hanks and McDermott, 1987] S. Hanks and D. McDermott. Nonmonotonic logic and temporal projection. *Artif. Intell.*, 33(3):379–412, 1987.

[Hölldobler and Schneeberger, 1990] S. Hölldobler and J. Schneeberger. A New Deductive Approach to Planning. *New Generation Computing*, 8:225–244, 1990.

[Hölldobler and Thielscher, 1995] S. Hölldobler and M. Thielscher. Computing Change and Specificity with Equational Logic Programs. *Annals of Mathematics and Artif. Intell.*, 1995.

[Hölldobler, 1989] S. Hölldobler. *Foundations of Equational Logic Programming*, vol. 353 of *LNAI*. Springer, 1989.

[Kartha and Lifschitz, 1994] G. N. Kartha and V. Lifschitz. Actions with Indirect Effects. In J. Doyle, E. Sandewall, and P. Torasso, ed.'s, *Proc. of KR*, p. 341–350, Bonn, Germany, May 1994. Morgan Kaufmann.

[Lifschitz, 1990] V. Lifschitz. Frames in the Space of Situations. *Artif. Intell.*, 46:365–376, 1990.

[Lin and Reiter, 1994] F. Lin and R. Reiter. State constraints revisited. *J. of Logic and Computation*, 4(5):655–678, 1994.

[Marek and Truszczyński, 1994] W. Marek and M. Truszczyński. Revision specifications by means of revision programs. In C. MacNish, D. Peirce, and L. M. Peireira, ed.'s, *Proc. of the European Workshop on Logics in AI (JELIA)*, vol. 838 of *LNAI*. Springer, 1994.

[Marek and Truszczyński, 1995] W. Marek and M. Truszczyński. Revision programming, database updates and integrity constraints. In *Proc. of ICDT*, p. 368–382. Springer, 1995.

[Masseron *et al.*, 1990] M. Masseron, C. Tollu, and J. Vauzielles. Generating Plans in Linear Logic. In *Foundations of Software Technology and Theoretical Computer Science*, vol. 472 of *LNCS*, p. 63–75. Springer, 1990.

[McCarthy and Hayes, 1969] J. McCarthy and P. J. Hayes. Some Philosophical Problems from the Standpoint of Artificial Intelligence. *Machine Intell.*, 4:463–502, 1969.

[Shepherdson, 1992] J. C. Shepherdson. SLDNF-Resolution with Equality. *J. of Autom. Reasoning*, 8:297–306, 1992.

[Thielscher, 1994] M. Thielscher. Representing Actions in Equational Logic Programming. In P. Van Hentenryck, ed., *Proc. of ICLP*, p. 207–224, Santa Margherita Ligure, Italy, June 1994. MIT Press.

[Thielscher, 1995] M. Thielscher. On the Completeness of SLDENF-Resolution. *J. of Autom. Reasoning*, 1995.

[7] This example is due to N. McCain and H. Turner.