Defeasible Specifications in Action Theories

Abstract

Recent research in formalizing effects of actions on a world in the presence of constraints have mostly concentrated on non-defeasible specifications, where the effect of actions and constraints are strictly specified. The focus of these research was more on solving the frame problem (and other problems such as the qualification and the ramification problem). In this paper we show how to incorporate defeasibility in the specification and at the same time solve the frame problem, and the other problems. In our approach we consider extensions of the high level language 'A' of Gelfond and Lifschitz and introduce defeasible constraints and effect propositions. While direct semantics of 'A' does not need a logical language, our semantics of the transition function (i.e., the effect of a single action) is defined using extended logic programming. This is due to the defeasibility of the specification. We then add additional logic programming rules to the logic programming characterization of the transition function to reason about the effect of a sequence of actions.

Content Areas: knowledge representation, temporal reasoning, common sense reasoning, nonmonotonic reasoning.

Abstract ID: A837

Declaration: This paper has not already been accepted by and is not currently under review for a journal or another conference, nor will it be submitted for such during IJCAI's review period.

1 Introduction and Motivation

Recent research in formalizing effects of actions on a world in the presence of constraints have mostly concentrated on non-defeasible specifications, where the effect of actions and constraints are strictly specified[1]. The focus of these research was more on solving the frame problem (and other problems such as the qualification and the ramification problem), and it made sense to start with non-defeasible specifications.

In this paper we show how to incorporate defeasibility in the specification and at the same time solve the frame problem, and the other problems. It is important to consider defeasible specifications for the same reason as the necessity of non-monotonic theories in knowledge representation and common-sense reasoning. The specifications are not definite and un-modifiable and may evolve. Our language should allow easy evolution or in the words of Mc Carthy, they should be 'elaboration tolerant'. We take a first step in this direction by allowing defeasible specifications.

In our approach we consider[2] extensions of the high level language $\mathcal{A}$ of Gelfond and Lifschitz [GL93] and introduce defeasible constraints and effect propositions. Direct semantics of $\mathcal{A}$ and many of its early successors (particularly, the formalization of transition functions that map state-action pairs to states) do not use a logical language. But our formalization of the transition function is done using extended logic programming. This is due to the defeasibility of the specification. (Earlier, Baral [Bar95] and Mc Cain and Turner [MT95] use logic programming and inference rules respectively, to formalize transition functions in presence of causal constraints.) We then add additional logic programming rules to the logic programming characterization of the transition function to reason about the effect of a sequence of actions.

The necessity of defeasible specifications may be illus-

[1] Among the various approaches, the strictness of the high level specification language of $\mathcal{A}$ [GL93] and its successors [KL94], and the language used by Sandewal [San92] is clear. Although, Reiter [Rei91] and his group do not use a particular specification language per se, the syntactic restriction they put on how the effect axioms and constraints can be initially specified in first-order logic, and the dependence of the compilation process that follows on the input syntax amounts to a specification language.

[2] We consider this language due to its simplicity. Our ideas are important for other specification languages and their formalizations. We would consider the impact of defeasible specifications on them in the full version of this paper.

trated by the following example from [Zha96]. Suppose we have a lamp with a switch on the side. *Turning the switch* causes an internal circuit in the lamp to flip between close (*not open*) and *open*. We can say then that the effect of executing the action *turning_switch* causes the circuit to close if it was *open*; or it will cause the switch to *open* if it was *not open*. It is also the case that when the circuit is close (*not open*) the *light* in the lamp is *on*, and when the circuit is *open* the light is *off*. Notice that although the state of the light is determined by the state of the circuit, the state of the light is indirectly affected by the action of turning the switch. This indirect effect is known in action theories as a *ramification* of the action. In an action description language such as AC [Tur95] this action domain may be described by the following propositions:[3]

$$turning\_switch \textbf{ causes } open \textbf{ if } \neg open \tag{1}$$
$$turning\_switch \textbf{ causes } \neg open \textbf{ if } open \tag{2}$$
$$open \textbf{ suffices for } \neg light\_on \tag{3}$$
$$\neg open \textbf{ suffices for } light\_on \tag{4}$$

Ramification effects are usually derived from the state of fluents,[4] while direct effects come from the execution of an action. An equivalent formulation can be written using Lin's formalization [Lin95] or the state specifications of Baral [Bar95].

Suppose now that we can also *unplug* the lamp and that a consequence of this action is that there will be no *power* going through the lamp. We can add to our description the following proposition to cover this action:

$$unplug\_lamp \textbf{ causes } \neg power \tag{5}$$

If there is no power going through the lamp the light in the lamp can never be on (this is regardless of the state of the circuit). We then must add the following ramification proposition:

$$\neg power \textbf{ suffices for } \neg light\_on \tag{6}$$

Notice too, this proposition together with the previous propositions may create contradictions or forbid certain sequences of actions. For example, executing the sequence [*unplug_lamp*, *turning_switch*] creates problems because there will be a state where the circuit is not *open* that causes the light to be on, and at the same time the light can not be on since the lamp is unplugged and there is no power going through the lamp. The problem is that a closed circuit not always, but *normally*, causes the light in lamp to be on.[5] Thus, the causality in proposition (4) must be *defeasible*.

In the following section we present $\mathcal{ADC}$, an action description language with defeasible causality propositions. To make it easier for the reader we have excluded many of the other features of action description languages such as constraints, that are described in the literature [Wor95]. Most of these can be easily added to $\mathcal{ADC}$.

2 The language $\mathcal{ADC}$

The language $\mathcal{ADC}$ has two disjoint non-empty sets of symbols called *fluents* and *actions*, and four kinds of *propositions*: *initial* propositions, *effect* propositions, *sufficiency* propositions and *defeasible sufficiency* propositions. An initial proposition is an expression of the form

$$\textbf{initially } f \tag{7}$$

where f is a fluent literal. A *fluent literal* is either a fluent or a fluent preceded by $\neg$. A fluent literal is *negative* if it is preceded by $\neg$, otherwise it is called positive. An effect proposition is an expression of the form

$$a \textbf{ causes } f \textbf{ if } p_1, \ldots, p_n \tag{8}$$

where a is an action, and f and and each of $p_1, \ldots, p_n$ $(0 \leq n)$ is a fluent literal. If $n = 0$ we will write:

$$a \textbf{ causes } f \tag{9}$$

Two effect propositions with preconditions $p_1, \ldots, p_n$ and $q_1, \ldots, q_m$ respectively are said to be *contradictory* if they describe the effect of the same action a on complementary fs, and $\{p_1, \ldots, p_n\} \cap \{\overline{q_1}, \ldots, \overline{q_m}\} = \emptyset$

Non-defeasible and defeasible sufficiency propositions are expressions of the form:

$$p_1, \ldots, p_n \textbf{ suffices for } f \tag{10}$$
$$p_1, \ldots, p_n \textbf{ normally suffices for } f \tag{11}$$

where f and each $p_1, \ldots, p_n$ $(1 \leq n)$ is a fluent literal. A *domain description* is a collection of initial, effect, sufficiency and defeasible sufficiency propositions. The initial propositions are referred to as *facts* and the other propositions are referred to as *causal laws*.

3 Transition functions of $\mathcal{ADC}$

We take a slightly different approach in defining the semantics of the language $\mathcal{ADC}$. Most of the action description languages [Tur95; KL94; BGP96] that are inspired by $\mathcal{A}$ have an independent characterization without involving any of the standard logical formalisms, such as logic programming, default logic, circumscription, classical logic, etc. But when we allow defeasible causality and keep open the possibility of hierarchies of such causality it is not clear if we can avoid using standard logical formalisms and have independent characterizations.

[3] At this point we appeal to the intuition of the reader with the meaning of the language. In the following sections will present a formal description of an action language similar to AC.

[4] Symbols that are used to represent the state of the world are called fluents. In the example the fluents are *open* and *light_on*.

[5] There are many other reasons why a closed circuit may not cause the light to be on. It could be that the bulb in the lamp is burned, or there was a storm that effected some power lines in the area and the entire neighborhood has no electricity.

An inherent component of semantics of action description languages is the transition function *Res* from actions and states to power set of states, where a *state* is a set of fluents. Intuitively, if $s' \in Res(a,s)$ we say that execution of action a in a situation with state s may take us to a situation with state s'. When we have only deterministic actions we can treat *Res* as a function from actions and states to states.

In this section we define the transition function of $\mathcal{ADC}$ through a translation of the domain description into an extended logic program. (We use the answer set semantics of extended logic programs [GL91].) Given a set of causal laws D, an action a, and a state s, we construct an extended logic program $\Pi(D,a,s)$, and use its semantics to define $Res(a,s)$. The extended logic program $\Pi(D,a,s)$ uses variables of three sorts: *situation* variables $S, S_1, \ldots$, *fluent* variables $F, F_1, \ldots$, and *action* variables $A, A_1, \ldots$.

For convenience, we use the following notations. The atom $Holds(f,s)$ denotes[6] the literal $holds(f,s)$ if f is a positive fluent literal and the literal $\neg holds(|f|,s)$ if f is a negative fluent literal. Also, for any fluent f, $|f| = f$ and $|\neg f| = f$. By $\overline{Holds(f,s)}$ we denote the literal which is complementary to $Holds(f,s)$. In addition, we will have an special postfix function symbol $'$ that we will apply to fluents and fluent variables. It is used to distinguish negative from positive fluent literals occurring as predicate arguments.[7]

We now describe the construction of the program $\Pi(D,a,s)$, which consists of the translations of the individual propositions in D and certain other rules.

Translation 3.1 *[Constructing* $\Pi(D,a,s)$*]*

1. Initial Database
If f is true in s then $\Pi(D,a,s)$ contains

(1.1) $holds(f,s) \leftarrow$

Otherwise, $\Pi(D,a,s)$ contains

(1.2) $\neg holds(f,s) \leftarrow$

2. Inertia Rule
$\Pi(D,a,s)$ contains the following inertia rules:

(2.1) $holds(F,res(a,s)) \leftarrow holds(F,s), \; not\ ab_0(F,a,s), not\ ab_1(F,a,s)$

(2.2) $\neg holds(F,res(a,s)) \leftarrow \neg holds(F,s), \; not\ ab_0(F',a,s), not\ ab_1(F',a,s)$

This rule is motivated by the minimality consideration which states that only changes that happens to a state as a result of an action a are the ones dictated by the direct and indirect effects of the action.

3. Translating Effect propositions

For each effect proposition of the form (8) in D which describes the effect of the action a, $\Pi(D,a,s)$ contains the following rules:

$$Holds(f,res(a,s)) \leftarrow Holds(p_1,s),\ldots,Holds(p_n,s)$$

$$ab_0(\overline{f},a,s) \leftarrow Holds(p_1,s),\ldots,Holds(p_n,s)$$

where $\overline{f} = l$ if $f = \neg l$; otherwise $\overline{f} = f'$.

4. Translating non-defeasible sufficiency propositions

For each non-defeasible sufficiency proposition of the form (10) in D, $\Pi(D,a,s)$ contains the following rules:

$$Holds(f,res(a,s)) \leftarrow Holds(p_1,res(a,s)),\ldots, Holds(p_n,res(a,s))$$

$$ab_0(\overline{f},a,s) \leftarrow Holds(p_1,res(a,s)),\ldots,Holds(p_n,res(a,s))$$

5. Translating defeasible sufficiency propositions

For each defeasible sufficiency proposition of the form (11) in D, $\Pi(D,a,s)$ contains the following rules:

$$Holds(f,res(a,s)) \leftarrow Holds(p_1,res(a,s)),\ldots, Holds(p_n,res(a,s)), not\ ab_0(f,a,s)$$

$$ab_1(\overline{f},a,s) \leftarrow Holds(p_1,res(a,s)),\ldots, Holds(p_n,res(a,s)), not\ ab_0(\overline{f},a,s)$$

□

Definition 3.1 For a situation s, and an action a, $Res(a,s)$ is defined with respect to the causal laws D through the program $\Pi(D,a,s)$ in the following way:

Let $\mathbf{A}$ be the set of answer sets of $\Pi(D,a,s)$.

If $\Pi(D,a,s)$ is inconsistent (i.e., only the set Lit is in $\mathbf{A}$) then $Res(a,s)$ is undefined, meaning that the action a is not executable in the state s.

Otherwise,

$Res(a,s) = \{\{f : holds(f,res(a,s)) \in A\} \cup \{\neg f : \neg holds(f,res(a,s)) \in A\} | A \in \mathbf{A}\}$ □

The structure of the translation presented above allows us to easily add more levels of defeasible propositions. If we ignore this issue then the translation is equivalent to a simpler version[8] where, we replace both ab_0 and ab_1 by ab, and in the first rule of step 5 we replace $not\ ab_0(f,A,s)$ by $not\ \overline{Holds(f,res(A,S))}$.

We would like to note that Turner [Tur95] uses $not\ \overline{Holds(f,res(A,S))}$ instead of an abnormal predicate. With this translation there is no obvious way to add defeasible causality to his formalism.

[6] The atom $p\_Holds(f,s)$, which will be used in a later section, denotes the literal $p\_holds(f,s)$ if f is a positive fluent literal and the literal $p\_holds(\overline{f},s)$ if f is a negative fluent literal.

[7] We could have used $\neg$ instead of $'$ but the unary negation $\neg$ is already used as predicate negation in extended logic programs.

[8] This was suggested by a colleague. We temporarily withhold the name to maintain our anonymity.

In general, when there are several levels of defeasibility the guard *not* $\overline{Holds(f, res(A, S))}$ can be used in the last (meaning highest) but one layer. All layers below that need an *ab* predicate. In the language $\mathcal{ADC}$ the lowest layer corresponds to inertia, the next layer corresponds to the defeasible causality and the highest layer is the effect propositions and non-defeasible causality propositions.

Note that $Res(a, s)$ is defined if contradictory effect rules are absent and sufficiency propositions are signed[9] In addition stratification of the sufficiency propositions guarantee $Res(a, s)$ to be a singleton set.

3.1 Examples

Example 3.1 Let D_1 consist of the the causal laws (1), (2), (3), and (4). Let $s_1 = \{open, power\}$, and $s_2 = \{light\_on, power\}$. Intuitively, the action *turning_switch* should cause the transition from s_1 to s_2 and from s_2 to s_1. We now verify that this indeed is the case, by examining the answer sets of the programs $\Pi(D_1, turning\_switch, s_1)$ and $\Pi(D_1, turning\_switch, s_2)$. For s_1 the initial database is:

$$S = \{holds(open, s_1), holds(power, s_1),\\ \neg holds(light\_on, s_1)\}$$

The following four rules are the effect propositions:

$$holds(open, res(turning\_switch, s_1)) \leftarrow \neg holds(open, s_1)$$
$$ab_0(open', turning\_switch, s_1) \leftarrow \neg holds(open, s_1)$$
$$\neg holds(open, res(turning\_switch, s_1)) \leftarrow holds(open, s_1)$$
$$ab_0(open, turning\_switch, s_1) \leftarrow holds(open, s_1)$$

Non-defeasible rules:

$$\neg holds(light\_on, res(A, s_1)) \leftarrow holds(open, res(A, s_1))$$
$$ab_0(light\_on', A, s_1) \leftarrow holds(open, res(A, s_1))$$
$$holds(light\_on, res(A, s_1)) \leftarrow \neg holds(open, res(A, s_1))$$
$$ab_0(light\_on', A, s_1) \leftarrow \neg holds(open, res(A, s_1))$$

Inertia rules:

$$holds(F, res(turning\_switch, s_1)) \leftarrow holds(F, s_1),\\ not\ ab_0(F, turning\_switch, s_1),\\ not\ ab_1(F, turning\_switch, s_1)$$
$$\neg holds(F, res(turning\_switch, s_1)) \leftarrow \neg holds(F, s_1),\\ not\ ab_0(F', turning\_switch, s_1),\\ not\ ab_1(F', turning\_switch, s_1)$$

The effect rules depend on the initial state S which will be part of any answer set of the program. Hence, the set

$$E = \{\neg holds(open, res(turning\_switch, s_1)),\\ ab_0(open, turning\_switch, s_1)\}$$

will also be part of any answer set. Therefore, from the third and fourth non-defeasible rules we get that the set:

$$ND = \{holds(light\_on, res(turning\_switch, s_1)),\\ ab_0(light\_on', turning\_switch, s_1)\}$$

will be part of any answer set too. The inertia rules, based on the initial state, will try to make the fluent literals *power*, $\neg light\_on$ and *open* true in the situation $res(turning\_switch, s_1)$. We can ignore the ab_1 predicates since there are no defeasible rules. However, ab_0 in E blocks *open* and ab_0 in ND blocks $\neg light\_on$. Hence, the inertia rules generate the set

$$I = \{holds(power, res(turning\_light, s_1))\}$$

No other *holds* literal with situation argument $res(turning\_light, s_1)$ or ab_0 predicate with third argument s_1 will be in any answer set. Notice too, that the evaluation of the rules was stratified, first the effect rules, then the non-defeasible rules and finally the inertia rules. Hence, it is easy to see that $Res(turning\_switch, s_1) = s_2$ and it is completely determined by $E \cup ND \cup I$.

Similarly, it can be shown that $Res(turning\_switch, s_2) = s_1$.

□

Example 3.2 If we add to the previous example the causal laws (5) and (6), we can see that the answer set of $\Pi(D_1, unplug, s_2)$ is inconsistent. This is because we will have the effect rule:

$$\neg holds(power, res(unplug, s_2)) \tag{12}$$

that is unconditionally true, and the following two non-defeasible sufficiency rules:

$$\neg holds(light\_on, res(unplug, s_2)) \leftarrow\\ \neg holds(power, res(unplug, s_2))$$
$$holds(light\_on, res(unplug, s_2)) \leftarrow\\ \neg holds(open, res(unplug, s_2))$$

The first rule will justify $\neg holds(light\_on, res(unplug, s_2))$ because of the effect rule and the second rule will justify $holds(light\_on, res(unplug, s_2))$ based on the inertia rule:

$$\neg holds(open, res(unplug, s_2)) \leftarrow \neg holds(open, s_2),\\ not\ ab_0(open', unplug, s_2),\\ not\ ab_1(open', unplug, s_2)$$

The literal $\neg holds(open, s_2)$ is part of the initial database, $open'$ will never appear in the first argument of any ab_0 in the translation and ab_1 is irrelevant in the example. □

Example 3.3 In this final example, we show the correct representation of the problem. We need the causal laws (1), (2), (5) and (6), plus the defeasible sufficiency propositions:

open **normally suffices for** $\neg light\_on$
$\neg open$ **normally suffices for** $light\_on$

In this case the second non-defeasible sufficiency rule from the previous example is replaced by the rule

$$holds(light\_on, res(unplug, s_2)) \leftarrow\\ \neg holds(open, res(unplug, s_2)),\\ not\ ab_0(ligt_on, unplug, s_2)$$

[9] We use the term 'signed' loosely. What we mean is that the sufficiency propositions result in a signing of the rules obtained in step 4 and 5. Stratification of sufficiency propositions has a similar meaning.

and this rule will not be justified since the following rule is also part of the translation

$ab_0(light\_on, unplug, s_2) \leftarrow \neg holds(power, res(unplug, s_2))$

which is justified by (12). □

4 Semantics of $\mathcal{ADC}$ and a logic programming translation

In this section we use the formalization of Res to present a semantics of $\mathcal{ADC}$ which allows us to make inferences about the effect of a sequence of actions.

An *interpretation* is a partial function Ψ from sequences of actions to states such that:

(1) The empty sequence [ ] belongs to the domain of Ψ and

(2) Ψ is prefix-closed [10].

$\Psi([\,])$ is called the initial state of Ψ.

Given a fluent f and a state σ, we say that f *holds* in σ (f is *true* in σ) if $f \in \sigma$; $\neg f$ *holds* in σ (f is *false* in σ) if $f \notin \sigma$.

Definition 4.1 An interpretation Ψ *satisfies* the causal laws of D if for any sequence $\alpha \circ a$ from the language of D,

if $Res(a, \Psi(\alpha))$ is defined

then

$\Psi(\alpha \circ a) \in Res(a, \Psi(\alpha))$,

otherwise

$\Psi(\alpha \circ a)$ is undefined.

We say Ψ is a *causal model* of D if it satisfies the causal laws of D. □

We say an initial proposition of the form **initially** f is true in an interpretation Ψ if f is true in $\Psi([])$.

Definition 4.2 An interpretation Ψ will be called a *model* of a domain description D in $\mathcal{ADC}$ if the following conditions are satisfied:

1. Ψ is a causal model of D.

2. Initial propositions of D are true in $\Psi([])$. □

A query in $\mathcal{ADC}$ is an expression of the form

$$f \textbf{ after } a_1, \ldots, a_n \tag{13}$$

where, f is a fluent literal and a_i's are actions.

Definition 4.3 We say a query of the form (13) is true in an interpretation Ψ if f is true in $\Psi([a_1, \ldots, a_n])$.

We say $D \models f$ **after** $a_1, \ldots, a_n$ if f is true in all models of $\Psi([a_1, \ldots, a_n])$. □

[10] By "prefix closed" we mean that for any sequence of actions α and action a, if $\alpha \circ a$ is in the domain of Ψ then so is α.

4.1 Logic programming characterization of $\mathcal{ADC}$

We can modify the translation $\Pi(D, a, s)$ to give a logic programming characterization of $\mathcal{ADC}$. We introduce a new predicate p_holds (which intuitively means 'possibly holds') and replace all occurrences of $holds(f, res(a, s))$ and $\neg holds(f, res(a, s))$ in the steps 1, 3, 4 and 5, and the head of the rules in step 2, by $p\_holds(f, res(a, S))$ and $p\_holds(f', res(a, S))$. We then modify step 2 by replacing the constants a and s, by the variables A and S. Then, we add the following rules:

$\neg reachable(S) \leftarrow p\_holds(F, S), p\_holds(F', S)$

$reachable(S) \leftarrow not\ \neg reachable(S)$

$holds(F, S) \leftarrow p\_holds(F, S), reachable(S)$

$\neg holds(F, S) \leftarrow p\_holds(F', S), reachable(S)$

And, finally we add rules to make sure that sufficiency propositions are true w.r.t. the initial state s_0. For a non-defeasible sufficiency proposition of the form (10) we add the rules:

$p\_Holds(f, s_0) \leftarrow p\_Holds(p1, s_0), \ldots, p\_Holds(p_n, s_0).$
$ab_{init}(\overline{f}, s_0) \leftarrow p\_Holds(p1, s_0), \ldots, p\_Holds(p_n, s_0).$

For defeasible sufficiency propositions of the form (11) we add the rule:

$p\_Holds(f, s_0) \leftarrow p\_Holds(p1, s_0), \ldots, p\_Holds(p_n, s_0), not\ ab_{init}(\overline{f}, s_0).$

Let us refer to the resultant program as $\Pi(D)$.

Proposition 4.1 For any domain description D with complete information about the initial state

$D \models f$ **after** $a_1, \ldots, a_n$ iff $\Pi(D) \models holds(f, [a_1, \ldots, a_n])$
□

5 Defeasible effect propositions

Our new approach of defining the semantics of the action language as a logic program gives us the flexibility to directly add defeasible effect propositions to $\mathcal{ADC}$. Consider the following effect proposition.
$open\_faucet$ **causes** $water\_on\_sink$.

However, in the rare occasion when the pipe is blocked the action will not have any effect[11]. A better description for the effect proposition would be:

$open\_faucet$ **normally_causes** $water\_on\_sink$.

The constraint about the case when the pipe is blocked can then be specified as:

$pipe\_blocked$ **suffices for** $\neg water\_on\_sink$.

In general, we will call a defeasible effect proposition an expression of the form

$$a \textbf{ normally\_causes } f \textbf{ if } p_1, \ldots, p_n \tag{14}$$

[11] Note the similarity to McCarthy's example of the potato in the pipe.

where a is an action, and f and and each of $p_1, \ldots, p_n$ $(0 \leq n)$ is a fluent literal.

To describe the semantics of a domain description D with defeasible effect propositions we expand the program $\Pi(D, a, s)$ as follows.

6. Translating Defeasible Effect propositions

For each defeasible effect proposition of the form (14) in D which describes the effect of the action a, $\Pi(D, a, s)$ contains the following rules:

$$Holds(f, res(a, s)) \leftarrow Holds(p_1, s), \ldots, Holds(p_n, s), \\ not\ pre\_ab(\overline{f}, s)$$

$$ab_0(\overline{f}, a, s) \leftarrow Holds(p_1, s), \ldots, Holds(p_n, s), \\ not\ pre\_ab(\overline{f}, s)$$

We also need to add to the two rules of the translation of non-defeasible sufficiency propositions the following rule (step 4 of the original transformation):

$$pre\_ab(\overline{f}, res(a, s)) \leftarrow Holds(q_1, s), \ldots, Holds(q_m, s).$$

To expand the logic programming characterization of $\mathcal{ADC}$ domains we merely need to replace the $Holds$ in the rules of part (6) with the corresponding p_holds and proposition 4.1 holds for the new domain descriptions.

In the above formalization we assumed that defeasible effect propositions are overridden by non-defeasible causal rules and defeasible effect propositions do not contradict with non-defeasible effect propositions. A different formalization can be given when we modify the assumptions such as (i) allowing defeasible effect propositions to contradict with non-defeasible effect propositions and giving non-defeasible effect propositions a higher priority; (ii) giving effect propositions higher priority than sufficiency propositions. The new formalizations will mainly express the priority between the different abnormal predicates. We will discuss this in greater detail in the full paper.

6 Conclusion

In this paper we showed how to formalize defeasible causality in action theories. We used logic programming to encode the defeasible causalities. It is not clear to us if use of a logical formalism can be avoided (in the spirit of $\mathcal{A}$ [GL93]) while encoding defeasibility, particularly when we allow several level of defeasibility. In the full paper we will add additional examples involving several levels of defeasibility and discuss how defeasibility can be incorporated in other specification languages.

To make our point clear we avoided additional constructs and features. Most of these can be easily added to the proposed language, and we will need default logic (as in [Tur95]), instead of logic programming, to encode defeasibility when we allow fluent formulas instead of fluent literals.

References

[Bar95] C. Baral. Reasoning about Actions : Non-deterministic effects, Constraints and Qualification. In *Proc. of IJCAI 95*, pages 2017–2023, 1995.

[BGP96] C. Baral, M. Gelfond, and A. Provetti. Representing Actions: Laws, Observations and Hypothesis. *Journal of Logic Programming (to appear)*, 1996.

[GL91] M. Gelfond and V. Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, pages 365–387, 1991.

[GL93] M. Gelfond and V. Lifschitz. Representing actions and change by logic programs. *Journal of Logic Programming*, 17(2,3,4):301–323, 1993.

[KL94] G. Kartha and V. Lifschitz. Actions with indirect effects: Preliminary report. In *KR 94*, pages 341–350, 1994.

[Lin95] F. Lin. Embracing causality in specifying the indirect effects of actions. In *Proc. of IJCAI 95*, pages 1985–1993, 95.

[MT95] N. McCain and M. Turner. A causal theory of ramifications and qualifications. In *Proc. of IJCAI 95*, pages 1978–1984, 95.

[Rei91] R. Reiter. The frame problem in the situation calculus: A simple solution (sometimes) and a completeness result for goal regression. In V. Lifschitz, editor, *Artificial Intelligence and Mathematical Theory of Computation*, pages 359–380. Academic Press, 1991.

[San92] E. Sandewall. Features and fluents: A systemetic approach to the representation of knowledge about dynamical systems. Technical report, Institutionen for datavetenskap, Universitetet och Tekniska hogskolan i Linkoping, Sweeden, 1992.

[Tur95] H. Turner. Representing actions in default logic: A situation calculus approach. In *Proceedings of the Symposium in honor of Michael Gelfond's 50th birthday (also in Common Sense 96)*, 1995.

[Wor95] Working notes of AAAI Spring Symposium. *Extending Theories of Action: Formal Theory and Practical Applications.* AAAI Press, 1995.

[Zha96] Y. Zhang. Compiling causality into action theories. In *Proc. of Common Sense 96*, pages 263–270, 1996.