Dependent Fluents

Enrico Giunchiglia
DIST - Università di Genova
Via Opera Pia 13, 16145 Genova, Italy
Email: enrico@dist.unige.it

Vladimir Lifschitz
Department of Computer Sciences
University of Texas, Austin, TX 78712, USA
Email: vl@cs.utexas.edu

Abstract

We discuss the persistence of the indirect effects of an action—the question when such effects are subject to the commonsense law of inertia, and how to describe their evolution in the cases when inertia does not apply. Our model of nonpersistent effects involves the assumption that the value of the fluent in question is determined by the values of other fluents, although the dependency may be partially or completely unknown. This view leads us to a new high-level action language $\mathcal{ARD}$ (for Actions, Ramifications and Dependencies) that is capable of describing both persistent and nonpersistent effects. Unlike the action languages introduced in the past, $\mathcal{ARD}$ is "non-Markovian," in the sense that the evolution of the fluents described in this language may depend on their history, and not only on their current values.

1 Introduction

This paper is about the *ramification problem* in the theory of commonsense reasoning, that is, about the problem of determining the indirect effects of an action. More specifically, we are interested in what Myers and Smith [1988] called the *persistence of derived information*—in deciding whether the indirect effects of an action should be presumed to persist, or, in other words, whether they are subject to the commonsense law of inertia.

Compare two examples:

Example 1 [Crawford, 1994]. If you are in the lake then you are wet. Jumping in the lake has an indirect effect—getting wet.

Example 2 [Myers and Smith, 1988]. If an object is on the table then it is not dangerous for the baby crawling on the floor. Putting an object on the table has an indirect effect—making it safe.

The examples look similar. Consider, however, what happens in each case if the action under consideration is followed by an action with the opposite effect. After you get out of the lake, you are still wet; this conclusion can be justified by the commonsense law of inertia. After an object is removed from the table, there is no guarantee that it is still safe from baby. Moreover, if we know that originally the object was not safe, and that now it is returned to the old location, then common sense tells us that it is definitely not safe. Somehow, the law of inertia does not apply.

The ramifications similar to Example 1 are by now well understood. Formal accounts of this case of the ramification problem, in various contexts, were given in [Ginsberg and Smith, 1988], [Winslett, 1988], [Baker and Ginsberg, 1989], [Baker, 1991], [Lifschitz, 1991], [Lin and Shoham, 1991], [Kartha and Lifschitz, 1994]. In the more difficult case of "noninertial ramifications," illustrated by Example 2, little progress has been made. Myers and Smith [1988] identified the problem, gave a few instructive examples and sketched an approach based on default logic. But their paper does not contain an actual formalization of any of the examples.

In Section 2, we present a new informal analysis of noninertial ramifications. This discussion leads us, in Section 3, to a high-level syntax, similar to the one used in [Gelfond and Lifschitz, 1993] and [Kartha and Lifschitz, 1994], that distinguishes between the two kinds of ramifications, and then, in Section 4, to an action language that allows us to formalize both Example 1 and Example 2. The new language extends the action language $\mathcal{AR}$ from [Giunchiglia *et al.*, 1994] and is called $\mathcal{ARD}$—for Actions, Ramifications and Dependencies. The properties of $\mathcal{ARD}$ are investigated in Section 5.

2 Dependencies

There is nothing peculiar about the fact that some fluents are "inertial" and others are not. For example, the law of inertia is often restricted to "primitive fluents" and is not applied to their propositional combinations in [Myers and Smith, 1988] and [Baker, 1991]. Several examples when fluents change "by themselves" are given in [Lifschitz and Rabinov, 1989]. In [Lifschitz, 1990], a subset of fluents is designated as a "frame," and a fluent is assumed to be inertial only if it belongs to the frame.

But if we simply declare the fluent $Safe(x)$ noninertial, then *any* action will be able to affect its value, even the trivial action $Wait$. Common sense tells us that this fluent can be affected by an action if that action affects the location of x, but not otherwise.

It appears that $Safe(x)$ is similar in its behavior to the noninertial fluents that have an explicit definition in terms of the location of x, except that *we do not know what this definition is.* If the top of the table is one of n possible locations of x then we are free to assume about any of the other $n-1$ locations that, being there, x would or would not be safe. There are 2^{n-1} possible "functional dependencies" between $Safe(x)$ and $Location(x)$, and we seem to implicitly assume that one of them holds, without specifying which one.

In connection with a similar example, Crawford [1994] writes:

> If we want derived consequences not to persist then we probably have to rethink our approach to developing semantics. The problem is that the persistence of fluents now seems to depend on their history—how they came to have the value they now have. As far as I know no current approaches allow this.

Indeed, traditional approaches to the semantics of actions have a "Markov property"[1] which can be informally described as follows: If the current values of all fluents mentioned in the problem are completely specified then no additional information about the values of these fluents in the past will allow us to make additional predictions about their future values. (This concept will be made precise in Section 6.) In Example 2, assume that all objects under consideration are currently on the table, and consequently safe. Then we know the current values of all fluents $Location(x)$ and $Safe(x)$. There is no way to decide on the basis of this information whether $HeavyHammer$ will be safe after we put it on the floor. But if we assume, in addition, that at some point in the past $HeavyHammer$ was on the floor and was not safe, then we can predict that the fluent $Safe(HeavyHammer)$ is going to change its value. We see that a formalism with the Markov property would not allow us to represent the $Safe$ example.

The idea of a (possibly unknown) dependency outlined above seems promising because it can give rise to "non-Markovian" formalisms. From this perspective, the state vector of an action domain includes both "explicit" components—the values of the fluents available in the language—and an "implicit" part, the dependencies that characterize the dependent fluents. This implicit part of the state vector does not change as the actions are performed, but it can affect the values of the transition function of the system. Additional information about the past can be used to learn the values of the implicit components, and consequently to arrive at new predictions.

3 Formalization of the Examples

Example 1 can be easily represented in the language $\mathcal{AR}$ [Giunchiglia *et al.*, 1994], or even in its dialect $\mathcal{AR}_0$ from [Kartha and Lifschitz, 1994]. An $\mathcal{AR}_0$ language is characterized by a set of "action names," such as $JumpIn$ and $JumpOut$, and a set of "fluent names," such as $InLake$ and Wet. Some fluent names are designated as "inertial" (or forming a "frame"); in this example, both $InLake$ and Wet are inertial.

[1] This terminology is suggested by a vague analogy with Markov chains in the theory of probability.

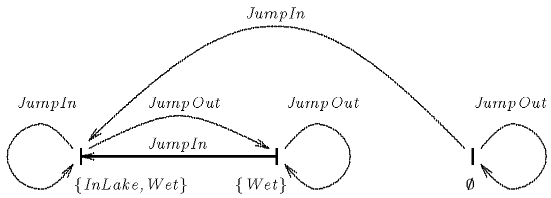

Figure 1: The transition diagram for Example 1.

The condition *if you are in the lake then you are wet* and the effects of the actions can be represented by the following "domain description":

$$\begin{array}{l} \textbf{always } InLake \supset Wet, \\ JumpIn \textbf{ causes } InLake, \\ JumpOut \textbf{ causes } \neg InLake. \end{array} \tag{1}$$

The first line of (1) is a "constraint"; the other two lines are "effect propositions."

According to the semantics of $\mathcal{AR}_0$, description (1) represents a certain transition system. The input symbols of the system are the action names $JumpIn$ and $JumpOut$. The states of the system are the valuations—functions from fluent names to truth values—that satisfy the formula $InLake \supset Wet$. If we agree to represent a valuation by the set of fluents to which it assigns the value $True$ then the set of states can be written as

$$\{\{InLake, Wet\}, \{Wet\}, \emptyset\}. \tag{2}$$

The transition diagram of this system is depicted in Figure 1.

Besides constraints and effect propositions, $\mathcal{AR}_0$ has also "value propositions" that represent conditions on the values of fluents in specific situations, such as

$$\begin{array}{l} InLake \wedge Wet \textbf{ after } JumpIn, \\ \neg InLake \wedge Wet \textbf{ after } JumpIn; JumpOut. \end{array} \tag{3}$$

For instance, the second value proposition expresses that performing the sequence of actions $JumpIn; JumpOut$ in the initial state will bring the system to a state which satisfies $\neg InLake \wedge Wet$. Both value propositions shown above are "entailed" by domain description (1).

In order to formalize Example 2, the language $\mathcal{AR}_0$ needs to be extended. We will introduce, besides inertial fluent names, a second special category, "dependent" fluent names. Thus, in the extended language, a fluent name can be designated as inertial or dependent, but not both. The language $\mathcal{AR}_0$ corresponds to the special case when there are no dependent fluent names.

For simplicity, we drop the argument in $Safe(x)$ and consider only two locations—the top of the table and the rest of the world. The action names are $PutOnTable$ and $RemoveFromTable$. The fluent names are $OnTable$ and $Safe$; $OnTable$ is inertial and $Safe$ is dependent. The domain description consists of the following propositions:

$$\begin{array}{l} Safe \textbf{ depends on } OnTable, \\ \textbf{always } OnTable \supset Safe, \\ PutOnTable \textbf{ causes } OnTable, \\ RemoveFromTable \textbf{ causes } \neg OnTable. \end{array} \tag{4}$$

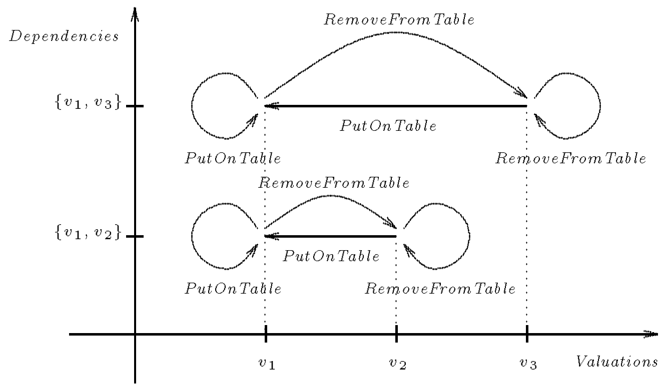

Figure 2: The transition diagram for Example 2.

The first line of (4) is a proposition of a new kind, a "dependency proposition."[2] The fact that the domain description includes this proposition and no other dependency propositions beginning with *Safe* tells us that the value of this fluent in any situation is determined by the value of the fluent *OnTable*. This assumption is nonmonotonic, in the sense that it can be defeated by adding other dependency propositions to (4).

In the extended language, a "state" is a pair. Its "explicit" component is what is considered a state in $\mathcal{AR}_0$—a valuation that satisfies the constraints of the given domain description. The "implicit" component is a "dependency," *i.e.* a set of valuations which are considered to be possible. For instance, the valuations that satisfy $OnTable \supset Safe$ are

$$\begin{aligned} v_1 &= \{OnTable, Safe\}, \\ v_2 &= \{Safe\}, \\ v_3 &= \emptyset. \end{aligned}$$

According to the semantics of the extended language described below, two dependencies can be used in this case as the implicit components of states. The dependency $\{v_1, v_2\}$ represents the case of a doll, which is safe no matter where it is located. The dependency $\{v_1, v_3\}$ represents a hammer, which is safe only when it is on the table. We will see that domain description (4) has the following states:

$$\begin{aligned} \sigma_1 &= \langle v_1, \{v_1, v_2\}\rangle, \\ \sigma_2 &= \langle v_2, \{v_1, v_2\}\rangle, \\ \sigma_3 &= \langle v_1, \{v_1, v_3\}\rangle, \\ \sigma_4 &= \langle v_3, \{v_1, v_3\}\rangle. \end{aligned} \tag{5}$$

The difference between σ_1 and σ_3 is the difference between a doll and a hammer that are out of reach of the baby. The explicit components of these states are the same, but their dependencies are different.

The transition diagram of this system is depicted in Figure 2. Note that the system is deterministic: Executing any action in any state leads to a uniquely determined state. However, the values of the fluents *OnTable* and *Safe* do not determine the state uniquely, and consequently do not predetermine the values of the fluents in the next state.

[2]Dependency propositions are somewhat similar to the formulas $INESSENTIAL(p, q)$ used in [Myers and Smith, 1988]. The authors' intention was to express "that fluent q is inessential in any justification of fluent p."

4 The Action Language $\mathcal{ARD}$

In this section, we assume that the reader is familiar with the description of the language $\mathcal{AR}_0$ in [Kartha and Lifschitz, 1994]. About the language $\mathcal{AR}$ [Giunchiglia *et al.*, 1994] it suffices to know that it differs from $\mathcal{AR}_0$ in two ways. First, fluents in $\mathcal{AR}$ do not have to be propositional; their values can come from any domains. Second, "release" propositions are replaced in $\mathcal{AR}$ by "indeterminate effect propositions" whose syntax is similar, but the semantics is somewhat different and is better suited for the task of formalizing nondeterministic actions.

4.1 Syntax

An $\mathcal{ARD}$ language is characterized by

- a nonempty set of symbols, that are called *fluent names*,
- a function, associating with every fluent name F a nonempty set Dom_F of symbols, that is called the *domain* of F,
- two disjoint subsets of fluent names; the elements of one are called *inertial*, and the elements of the other *dependent*,
- a nonempty set of symbols, that are called *action names*.

An *independent* fluent name is any fluent name that is not dependent; in particular, all inertial fluent names are independent.

An *atomic formula* is an expression of the form

$$(F \textbf{ is } V)$$

where F is a fluent name and $V \in Dom_F$. A *formula* is a propositional combination of atomic formulas.

There are five types of *propositions* in $\mathcal{ARD}$—value propositions, constraints, determinate and indeterminate effect propositions, and dependency propositions. A *value proposition* is an expression of the form

$$C \textbf{ after } \overline{A} \tag{6}$$

where C is a formula, and $\overline{A}$ is a string of action names. A *constraint* is an expression of the form

$$\textbf{always } C \tag{7}$$

where C is a formula. A *determinate effect proposition* is an expression of the form

$$A \textbf{ causes } C \textbf{ if } P \tag{8}$$

where A is an action, and C and P are formulas. An *indeterminate effect proposition* is an expression of the form

$$A \textbf{ possibly changes } F \textbf{ if } P \tag{9}$$

where A is an action name, F an inertial fluent name, and P a formula. Finally, a *dependency proposition* is an expression of form

$$F \textbf{ depends on } F' \textbf{ if } P \tag{10}$$

where F is a dependent fluent name, F' any fluent name, and P a formula.

A *domain description* is a set of propositions.

The following abbreviations will be used. In formulas, some parentheses will be omitted, as customary in propositional logic. If $\overline{A}$ in a value proposition (6) is empty, we will write this proposition as

$$\textbf{initially } C.$$

Otherwise, the members of $\overline{A}$ will be separated by semicolons. In effect propositions (8), (9) and in dependency propositions (10), the part **if** P will be dropped if P is the propositional formula $True$. A fluent F is *propositional* if $Dom_F = \{False, True\}$; for a propositional fluent F, we will abbreviate the atomic formula

$$F \textbf{ is } True$$

by F.

With these notational conventions, (1) and (4) become domain descriptions in the language $\mathcal{ARD}$. The following enhanced formalization of Example 2 illustrates the use of nonpropositional fluents. Let $L_1, \ldots, L_n$ $(n > 1)$ be distinct symbols. Replace the propositional fluent $OnTable$ in the language of (4) by the fluent $Location$, whose domain is $\{L_1, \ldots, L_n\}$; $OnTable$ will be treated as an alternative notation for L_1. The new domain description is

$$\begin{array}{l} Safe \textbf{ depends on } Location, \\ \textbf{always } (Location \textbf{ is } OnTable) \supset Safe, \\ PutOnTable \textbf{ causes } (Location \textbf{ is } OnTable), \\ RemoveFromTable \textbf{ causes } \neg(Location \textbf{ is } OnTable). \end{array} \quad (11)$$

If $n = 2$ then (11) is essentially the same as (4).

Here is a further enhancement, illustrating the use of P in dependency propositions (10). We would like to express that the closet is a safe location if its door is closed. Specifically, if the object is in the closet, then its safety depends on whether or not the door is closed; if it is then the object is definitely safe. This can be done by extending (11) as follows. We add to the language the inertial propositional fluent name $DoorClosed$ and the action names $CloseDoor$, $OpenDoor$. The following propositions are added to (11):

$$\begin{array}{l} Safe \textbf{ depends on } DoorClosed \textbf{ if } (Location \textbf{ is } InCloset), \\ \textbf{always } (Location \textbf{ is } InCloset) \wedge DoorClosed \supset Safe, \\ CloseDoor \textbf{ causes } DoorClosed, \\ OpenDoor \textbf{ causes } \neg DoorClosed \end{array}$$

($InCloset$ stands for L_2). In this new domain description, $Safe$ depends on $Location$ and $DoorClosed$ if the object is in the closet, and only on $Location$ otherwise. Opening the door may affect the safety of the object only if it is in the closet.

4.2 Semantics: Dependencies and States

A *valuation* is a function v defined on the set of fluent names such that, for every fluent name F, $v(F) \in Dom_F$. Any valuation v can be extended to atomic formulas as follows:

$$v(F \textbf{ is } V) = \begin{cases} True, & \text{if } v(F) = V, \\ False, & \text{otherwise.} \end{cases}$$

It can be further extended to arbitrary formulas according to the truth tables of propositional logic. A valuation v *satisfies* a constraint (7) if $v(C) = True$.

Consider a domain description D. A fluent F' is an F-*argument* for a valuation v if there is a dependency proposition (10) in D such that $v(P) = True$. A *dependency* is a set of valuations. Two valuations v and v' are said to be *incompatible* if there exists a dependent fluent F such that $v(F) \neq v'(F)$, the set of F-arguments for v is the same as for v' and for each F-argument F' we have $v(F') = v'(F')$. A dependency Δ is *coherent* if

- every valuation in Δ satisfies the constraints in D, and
- does not contain incompatible valuations.

It is clear that a subset of a coherent dependency is a coherent dependency also. We are interested in the "maximally strong" coherent dependencies, that is, in the coherent dependencies that are maximal with respect to set inclusion. For istance, the maximal coherent dependencies of (4) are $\{v_1, v_2\}$ and $\{v_1, v_3\}$.

A *state* of a domain description D is a pair $\langle v, \Delta \rangle$, where Δ is maximal among the dependencies coherent relative to D, and $v \in \Delta$. For instance, the states of (4) are $\sigma_1, \ldots, \sigma_4$ as defined by (5).

4.3 Semantics: The Transition Function

The key element of the semantics of $\mathcal{ARD}$ is the definition of the "result" functions Res^Δ, where Δ is a dependency. Each of these functions applies to an action name and an element of Δ, and produces a subset of Δ. Together, these functions define a nondeterministic transition function on the set of states, as follows: Given an action name A and a state $\langle v, \Delta \rangle$, the possible successor states have the form $\langle v', \Delta \rangle$, where $v' \in Res^\Delta(A, v)$.

For any dependency Δ, $Res_0^\Delta(A, v)$ is the set of valuations v' in Δ such that for each determinate effect proposition (8) in D, $v'(C) = True$ whenever $v(P) = True$. The elements of $Res^\Delta(A, v)$ will be the valuations in $Res_0^\Delta(A, v)$ which differs from v as little as possible. The difference between v and a valuation $v' \in Res_0^\Delta(A, v)$ is measured by $New_A(v, v')$, defined as the set of formulas

$$F \textbf{ is } v'(F) \quad (12)$$

such that

- F is inertial and $v(F) \neq v'(F)$, or
- for some indeterminate effect proposition (9) in D, $v(P) = True$.

The condition $v'(F) \neq v(F)$ expresses that (12) is a "new fact" that becomes true if the execution of A in a state $\langle v, \Delta \rangle$ results in state $\langle v', \Delta \rangle$. The set $New_A(v, v')$ includes such "new facts" for all inertial fluents F. (If F is noninertial then it is not expected to keep its old value after performing an action, so that the change in its value is disregarded.) On the other hand, if some indeterminate effect proposition allows F to change, we treat its value in $\langle v', \Delta \rangle$ as "new" even if it happens to coincide with the value of F in $\langle v, \Delta \rangle$.

Now we define $Res^{\Delta}(A, v)$ to be the set of valuations $v' \in Res_0^{\Delta}(A, v)$ for which $New_A(v, v')$ is minimal relative to set inclusion—in other words, for which there is no $v'' \in Res_0^{\Delta}(A, v)$ such that $New_A(v, v'')$ is a proper subset of $New_A(v, v')$.

It is easy to check that in the case of domains (1) and (4) this definition leads to the transition diagrams shown in Figures 1 and 2 respectively.

4.4 Semantics: Models and Entailment

A *structure* is a pair $\langle \Delta, \Psi \rangle$ where Δ is a dependency and Ψ is partial function from strings of actions into Δ whose domain is nonempty and prefix-closed. If Ψ is defined on a string $\overline{A}$, we say that $\overline{A}$ is *executable* in Ψ.

A value proposition C **after** $\overline{A}$ is *true* in a structure $\langle \Delta, \Psi \rangle$ if $\overline{A}$ is executable in Ψ and $\Psi(\overline{A})(C) = True$.

Consider a domain description D. A structure $\langle \Delta, \Psi \rangle$ is a *model* of D if Δ is maximal and coherent, every value proposition in D is true in Ψ, and for every string of actions $\overline{A}$ executable in Ψ and every action A one of the following two conditions is satisfied:

- $\overline{A}A$ is executable in Ψ and
$$\Psi(\overline{A}A) \in Res^{\Delta}(A, \Psi(\overline{A})),$$
or
- $\overline{A}A$ is not executable in Ψ and $Res^{\Delta}(A, \Psi(\overline{A})) = \emptyset$.

A value proposition is *entailed* by a domain description D if it is true in every model of D.

For instance, value propositions (3) are entailed by domain description (1) in the sense of this definition. Consider the corresponding propositions for the second example:

$$OnTable \wedge Safe \textbf{ after } PutOnTable,$$
$$\neg OnTable \wedge Safe \textbf{ after } PutOnTable; RemoveFromTable. \tag{13}$$

The first of them is entailed by domain description (4), but the second is not. If we add the "initial condition"

$$\textbf{initially } \neg OnTable \wedge Safe \tag{14}$$

to (4) then the extended domain description will entail both propositions in (13).

5 Some Properties of $\mathcal{ARD}$

It is not difficult to verify that, in the absence of dependent fluents, the semantics of $\mathcal{ARD}$ is equivalent to the semantics of $\mathcal{AR}$. In this case, the set of all states in the sense of $\mathcal{AR}$ is the only maximal coherent dependency.

Several properties of $\mathcal{AR}_0$ established in [Kartha and Lifschitz, 1994] can be proved for $\mathcal{ARD}$ as well. Clearly, adding a value proposition to a domain description can only make the set of its models smaller, and hence the set of value propositions entailed by it bigger (a "restricted monotonicity property" in the sense of [Lifschitz, 1993]). Any formula can be replaced by an equivalent formula without changing the set of models, and explicitly defined fluent names can be eliminated.

A *constant* in a domain description D is a dependent fluent F such that no dependency proposition in D begins with F. In any model of D, the value of a constant after executing any sequence of actions is the same as in the initial situation.

6 The Markov Property

Let $\overline{A}$ be a string of actions. We say that a domain description D is $\overline{A}$*-complete* if for every fluent F there exists a value $V \in Dom_F$ such that D entails

$$(F \textbf{ is } V) \textbf{ after } \overline{A}. \tag{15}$$

We say that two domain descriptions D_1 and D_2 are $\overline{A}$*-equivalent* if

- every proposition that belongs to one of the descriptions D_1, D_2 but not to the other is a value proposition
$$C \textbf{ after } \overline{A'}$$
such that $\overline{A}$ is not a prefix of $\overline{A'}$,
- for every fluent F and every value $V \in Dom_F$, proposition (15) is entailed by D_1 if and only if it is entailed by D_2.

The following theorem expresses the Markov property for $\mathcal{AR}$ domain descriptions (that is, for $\mathcal{ARD}$ domain descriptions without dependent fluents.)

Proposition. *Let D_1, D_2 be $\mathcal{AR}$ domain descriptions, and let $\overline{A}$, $\overline{A'}$ be strings of action names such that $\overline{A}$ is a prefix of $\overline{A'}$. If D_1 and D_2 are $\overline{A}$-complete and $\overline{A}$-equivalent then they are $\overline{A'}$-equivalent.*

For $\mathcal{ARD}$ domain descriptions, this is generally not true. For example, let D_1 be (4), and let D_2 be obtained from D_1 by adding value proposition (14). Take $\overline{A}$ to be $PutOnTable$, and take $\overline{A'}$ be $PutOnTable; RemoveFromTable$. It is easy to see that D_1 and D_2 are $\overline{A}$-complete and $\overline{A}$-equivalent, but not $\overline{A'}$-equivalent.

Acknowledgements

We are grateful to Michael Gelfond, Matthew Ginsberg, Neelakantan Kartha, Norman McCain, David Smith and Hudson Turner for useful discussions related to the subject of this paper. This work was partially supported by National Science Foundation under grant IRI-9306751.

References

[Baker and Ginsberg, 1989] Andrew Baker and Matthew Ginsberg. Temporal projection and explanation. In *Proc. of IJCAI-89*, pages 906–911, 1989.

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

[Crawford, 1994] James Crawford. Three issues in action. Unpublished note for the Fifth International Workshop on Nonmonotonic Reasoning, 1994.

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

[Ginsberg and Smith, 1988] Matthew L. Ginsberg and D.E Smith. Reasoning about action I: a possible worlds approach. *Artificial Intelligence*, 35:165–195, 1988.

[Giunchiglia *et al.*, 1994] Enrico Giunchiglia, G. Neelakantan Kartha, and Vladimir Lifschitz. Actions with indirect effects. Working notes of the AAAI Spring Symposium on Extending Theories of Action, 1994.

[Kartha and Lifschitz, 1994] G. Neelakantan Kartha and Vladimir Lifschitz. Actions with indirect effects (preliminary report). In *Proc. of the Fourth Int'l Conf. on Principles of Knowledge Representation and Reasoning*, pages 341–350, 1994.

[Lifschitz and Rabinov, 1989] Vladimir Lifschitz and Arkady Rabinov. Things that change by themselves. In *Proc. of IJCAI-89*, pages 864–867, 1989.

[Lifschitz, 1990] Vladimir Lifschitz. Frames in the space of situations. *Artificial Intelligence*, 46:365–376, 1990.

[Lifschitz, 1991] Vladimir Lifschitz. Towards a metatheory of action. In James Allen, Richard Fikes, and Erik Sandewall, editors, *Proc. of the Second Int'l Conf. on Principles of Knowledge Representation and Reasoning*, pages 376–386, 1991.

[Lifschitz, 1993] Vladimir Lifschitz. Restricted monotonicity. In *Proc. AAAI-93*, pages 432–437, 1993.

[Lin and Shoham, 1991] Fangzhen Lin and Yoav Shoham. Provably correct theories of action (preliminary report). In *Proc. AAAI-91*, pages 349–354, 1991.

[Myers and Smith, 1988] Karen Myers and David Smith. The persistence of derived information. In *Proc. AAAI-88*, pages 496–500, 1988.

[Winslett, 1988] Marianne Winslett. Reasoning about action using a possible model approach. In *Proc. AAAI-88*, pages 89–93, 1988.