Embracing Causality in Inducing the Effects of Actions

Ramon P. Otero

Department of Computer Science
University of Corunna
Corunna 15071, Galicia, Spain
otero@udc.es

Abstract. The following problem will be considered: from scattered examples on the behavior of a dynamic system induce a description of the system. For the induced description to be concise and modular, we use a generic action formalism based on causality, that is representable in logic programming. It is relatively simple to induce a description of a dynamic system that suffers from the frame problem. The known solutions to the frame problem require a non-monotonic formalism. Unfortunately induction under non-monotonic formalisms, e.g. normal logic programs, is not well understood yet. We present a method for induction under the non-monotonic behavior needed to solve the frame problem. Technically we introduce a causality predicate for the target fluent and induce a description of the causality of the fluent instead of the fluent itself. The description of causality together with the appropriate inertia axiom models the behavior of the original target fluent. The main advantage of this method is that the induction of the effects of actions can be made with well known induction methods on monotonic formalisms, such as Horn programs.

1 Introduction

Over the years, the problem of learning a description of a dynamic system was extensively considered, e.g., in the area of learning automata. Automata-based descriptions are clearly understood and have efficient algorithms for inference. But it is not easy to extend a description to cope with additional behaviors and the size of the description becomes very large for actual domains.

In this work we will consider action formalisms, an alternative approach for describing dynamic systems. From a point of view of automata, action formalisms provide a concise and highly modular description of the transition relation of the domain. Action formalims are logic-based, but unfortunately rely on non-monotonic logics.

The method of addressing this learning problem, started from Inductive Logic Programming (ILP). In this area of machine learning, a logic program is induced from examples of its conclusions. Some action formalisms can be represented in logic programs. Then induction of action descriptions can be solved with ILP.

R. Conejo et al. (Eds.): CAEPIA-TTIA 2003, LNAI 3040, pp. 291–301, 2004.

Other authors already followed this method [Moyle 2002, Lorenzo and Otero 2000] with restricted success. The reason is that ILP methods are only defined for definite logic programs, while action formalisms need normal programs to represent the nonmonotonic behavior using negation as failure (NAF). In definite logic programs, action descriptions have the frame problem (section 3).

The method followed in this work appeals to causality. Causality in action has provided solutions to the frame and ramification problems still to be solved in other formalisms. In induction, causality will provide a method to translate the nonmonotonic induction problem of the effects of actions to a form in which monotonic methods of ILP can be used, nevertheless providing a complete method for induction of action descriptions, and solving the frame problem in induction.

In the next section induction of the effects of actions is defined after a short introduction of action formalisms and ILP. Section 3 shows the presence of the frame problem in monotonic induction. The method of induction is presented in section 4 with some examples and characterization results. Then the method is extended (section 5) for its applicability to realistic domains. We conclude discussing the results and commenting on related work.

2 Induction of Action Descriptions

Every action description of a dynamic system distinguish between the evolving properties of the domain that are represented by *fluents*, and the *actions* that cause change in these properties. In logic programing, actions and fluents are represented as predicates. The steps through which the domain evolves are represented by a situation term, that is given as argument to every fluent $f(S)$, and action $a(S)$.

Then an action description, in logic programming, is a set of program rules with the following general form,

$$f(S) \leftarrow a(S),\ prev(S, PS),\ f'(PS), \ldots \tag{1}$$

Where f stands for a fluent and a for an action. The fluent atoms $f'(PS)$ are optional and always refer to the previous situation PS (assume the appropriate definition for predicate $prev(S, PS)$, e.g. $prev(2, 1)$). These rules are called *action laws* and describe the effects of performing action a at situation S as some fluent f being true at the same situation, under the *precondition* that other fluents $f'(PS)$ are true at the previous situation. Note that no reference far from the previous situation is allowed and that the rules cannot use constants in situation arguments, i.e. action laws hold for every situation.

Descriptions like this can be used to infer the effects of performing a sequence of actions from an initial state of the domain (*temporal prediction*). To this end a set of facts on the fluents at the initial situation, e.g. $f(0) \leftarrow$, $f'(0) \leftarrow$, and a corresponding set for the sequence of actions, e.g. $a(1) \leftarrow$, $a'(2) \leftarrow$, $a(3) \leftarrow$, is added to the program. The effects correspond to the consequences of the program on fluents at the different situations.

When inducing the effect of actions, the problem is somehow the opposite. From a set of ground instances of fluents and actions at the situations inside a sequence infer the corresponding set of rules. Following usual methods, we will define induction for one fluent at a time.

Note that the representation we introduced so far, does not allow for ground descriptions starting at different initial states (with different sets of ground fluent facts at situation 0) to be present simultaneously inside the program. Then we introduce another reference for the initial state that is also given as argument to every fluent and action. An action law will have the general form[1] $f(S,N) \leftarrow a(S,N),\ prev(S,PS),\ f'(PS,N),\ldots$ while a ground fact like $f(2,n1)$ will represent fluent f is true at situation 2 when starting from initial state $n1$.

Even in the case of fixed initial state, we cannot represent different sequences of actions in the program. Lets call *narrative* a particular sequence of actions starting from a particular initial state. The term we added before will be used to allow the representation of different narratives in the program.

Definition 1 (Induction of Effects of Actions) *Let*

i) A_i *be a set of ground facts on the actions at the situations in a narrative* i*,*
ii) F'_i *a corresponding set of ground facts on the fluents at the same narrative,*
iii) F_i^+ *(positive examples) a set of ground facts on some selected* target *fluent* f*, and*
iv) F_i^- *(negative examples) another set of ground facts on it.*

Given several collections of these four sets for different narratives, a set of rules P *in the form of action laws is a solution to induction of target fluent* f *if and only if, for every narrative* i*,*

$$(P \cup F'_i \cup A_i) \models F_i^+ \quad \textit{and} \tag{2}$$

$$(P \cup F'_i \cup A_i) \not\models F_i^- \tag{3}$$

The definition follows those of a general induction problem in ILP. From a set of positive examples E^+ and negative examples E^- on some predicate, and under some background Horn program B, a set of definite rules H is solution iff $B \cup H \models E^+$, $B \cup H \not\models E^-$, being also $B \cup H \not\models \bot$. The correspondence with induction in actions is $E^+ = \bigcup_i F_i^+$, $E^- = \bigcup_i F_i^-$, $B = \bigcup_i (A_i \cup F'_i)$, and $H = P$.

In particular we will use the method of Inverse Entailment (IE) [Muggleton 1995], for which an efficient implementation, called Progol, is available. IE and its implementation Progol, allows the specification of the intended form of the induced rules through the use of *mode declarations*. Search is then restricted to this bias. This is important for induction in actions, because any solution not in the form of an action law will not actually constitute a solution. Under some conditions, IE is a complete induction method [Muggleton 1998] for Horn programs providing every solution in the form of definite rules, this will be enough for the induction of the effect of actions.

[1] When it is clear from the context, we usually omit this additional reference to N to improve readability.

3 The Frame Problem in Induction

To show the existence of the frame problem in induction consider the simple well-known example of the Yale Shoting Scenario (YSS). There is a turkey and a gun, the gun can be loaded or not, and the turkey will be dead when shooting with the gun loaded. There are actions shoot s, load l, and wait w; and fluents loaded ld, and dead d. An example of a learning problem in this scenario is the induction of a description of fluent d from sets of examples on narratives like the following.

situation	0	1	2	3	4	5
dead	nd	nd	nd	nd	d	d
loaded	nld	nld	ld	ld	ld	ld
actions		s	l	w	s	w

Where nd (not dead) is the complementary fluent to d and nld the complementary of ld; recall that actions are represented at the same situation of their effects, instead of the previous one. A direct coding of this problem in the ILP system Progol can be as follows.

$$
\begin{array}{ll}
B & nld(0).\ nld(1).\ ld(2).\ ld(3).\ ld(4).\ ld(5).\ \ldots \\
 & s(1).\ l(2).\ w(3).\ s(4).\ w(5).\ \ldots \\
E^+ & d(4).\ d(5).\ \ldots \\
 & nd(0).\ nd(1).\ nd(2).\ nd(3).\ \ldots \\
E^- & :\!-\ d(0).\ :\!-\ d(1).\ :\!-\ d(2).\ :\!-\ d(3).\ \ldots \\
 & :\!-\ nd(4).\ :\!-\ nd(5).\ \ldots \\
\hline
H & d(S) :\!-\ s(S),\ prev(S,PS),\ ld(PS). \\
 & d(S) :\!-\ w(S),\ prev(S,PS),\ d(PS). \\
 & d(S) :\!-\ l(S),\ prev(S,PS),\ d(PS). \\
 & nd(S) :\!-\ w(S),\ prev(S,PS),\ nd(PS). \\
 & nd(S) :\!-\ l(S),\ prev(S,PS),\ nd(PS). \\
 & nd(S) :\!-\ s(S),\ prev(S,PS),\ nld(PS),\ nd(PS).
\end{array}
$$

The facts and constraints grouped under B, E^+ and E^- are the input to the system. The negative examples on E^- are represented as constraints. The rules grouped under H are the induced output by the Progol system. (Not shown is a ground description of predicate $prev(S, PS)$ also in B.)Looking at H, two kinds of rules have been induced, with the general forms:

$$f(S) \leftarrow a(S),\ prev(S,PS),\ f'(PS) \tag{4}$$

$$f(S) \leftarrow a(S),\ prev(S,PS),\ f(PS),\ f'(PS) \tag{5}$$

Where f stands for a fluent (d and nd) and a for an action, being f' a fluent different from f. Rules in the form (4) are (genuine) action laws, while those in the form (5) are called *frame axioms*. Note the number of frame axioms (five) compared with action laws (one).

Action laws explain the example instances when there is a change in truth, e.g. $d(4)$ true after $d(3)$ false. Frame axioms explain the example instances when

they persisted from the previous situation, note the head fluent $f(S)$ is also a condition of the rule at the previous situation $f(PS)$. Indeed all the rules are needed because all of the examples provided must be covered by H solution.

Every representation of actions must contain rules that describe persistent fluents after an action. It is the form of these rules – and the number of them – that states whether a representation has or not the frame problem. In the solution induced we have one frame axiom for each combination of fluent and action in the domain. Precisely because H contains so many frame axioms, the description induced has the frame problem. The frame problem is solved when the description of persistent fluents is made in a compact form, typically with a single rule for each fluent without referencing any action, or with a single rule for the domain, lets call these rules *inertia axioms*. For example, the rules $d(S) :- \ prev(S, PS),\ d(PS),\ not\ nd(S)$ and $nd(S) :- \ prev(S, PS),\ nd(PS),\ not\ d(S)$ would replace the five frame axioms.

Most of the methods of induction are defined on monotonic formalisms, then it can be said that every solution on them will have the frame problem. There are also some restricted nonmonotonic induction methods, but as far as our knowledge, their applicability on the frame problem has not been shown. Methods of nonmonotonic induction are still under study. It must be also said that as far as the induced model of the domain, solutions with the frame problem contain all the possible solutions. Solving the frame problem in induction will not provide additional models for the domain.

4 The Method

The method is based on the following. Frame axioms are induced because there are examples on fluents that persist. But we actually know the compact form for these rules, the inertia axioms, given the set of fluents. If the induction method were able to work under nonmonotonic normal programs, just making the inertia axioms present during induction in the background B, would provide solutions free from the frame problem.

Nevertheless there is a simple alternative, if frame axioms cover persistent examples, we can avoid their induction by not providing the persistent examples. In order to carefully make the selection of examples, we appeal to causality. A generic causal formalism of action will be used, that can be consider a summary of different causal approaches [Lin 1995,Gustafsson and Doherty 1996].

Step 1. For every fluent in the domain, define two predicates $f(S)$ and $nf(S)$, to represent when the fluent is true and when it is false. Add also the constraint

$$:- f(S), nf(S). \tag{6}$$

This representation avoids the CWA of LP for fluents and allows the reference to the negative fluent without using NAF, thus inside definite LP. The CWA for fluents is not interesting, because when some fluent instance cannot be proved it is better to assume it persisted than to assume it is false.

Step 2. For the target fluent $f(S)$ define an additional fluent $pf(S)$ (also $pnf(S)$ for $nf(S)$) as follows,

$$pf(S) \text{ :- } f(S),\ prev(S, PS),\ nf(PS). \tag{7}$$

$$pnf(S) \text{ :- } nf(S),\ prev(S, PS),\ f(PS). \tag{8}$$

Fluent $pf(S)$ is the causality predicate representing when the target fluent $f(S)$ is caused (and true). (Respectively $pnf(S)$ represents the target is caused and false.) From the set of examples on the target fluent f, these two rules will extract the corresponding set of examples on its causality. These two rules represent the initial idea of causality, namely, when there is a change in a fluent, the fluent must be caused.

Step 3. For the target fluent $f(S)$ define the fluent $npf(S)$ (also $npnf(S)$ for $nf(S)$) as follows,

$$npf(S) \text{ :- } nf(S). \tag{9}$$

$$npnf(S) \text{ :- } f(S). \tag{10}$$

Fluent $npf(S)$ is the complementary of $pf(S)$, i.e. represents the target fluent $f(S)$ is not caused. As in the previous step, from the set of examples, these two rules will extract the corresponding set on its non-causality. Note that non-causality is not defined from a persistent example, but instead from the complementary fluent being true. The fluent cannot be caused and true if it is false. Instead, if non-caused is defined from persistent, some solutions will be forbiden.

Step 4. Apply a complete monotonic induction method of ILP, e.g. IE, as in Definition 1 but with target fluent $pf(S)$ (then also for $pnf(S)$) instead of the original target $f(S)$. The instances of $pf(S)$ being the positive examples and those of $npf(S)$ the negative ones for the induction of $pf(S)$. (Respectively the instances on $pnf(S)$ and $npnf(S)$, are the positive and negative examples for $pnf(S)$.) The causality of the target fluent will be induced in the form of action laws:

$$pf(S) \text{ :- } a(S),\ prev(S, PS),\ f'(PS), \ldots \tag{11}$$

Note that the induced rules cannot correspond to frame axioms, simply because we did not provide positive examples on persistent fluents.

Step 5. To complete the solution add to the induced causal action laws (11) the inertia axiom for the original target $f(S)$, and a rule transfering causality to truth. (Also for the complementary $nf(S)$, not shown.)

$$f(S) \text{ :- } prev(S, PS),\ f(PS),\ not\, pnf(S). \tag{12}$$

$$f(S) \text{ :- } pf(S) \tag{13}$$

Recall the example on the YSS. We already used step 1 defining f and nf for each fluent. After steps 2 and 3 are applied we got the following extended description of the example narrative.

situation	0	1	2	3	4	5
dead	*nd*	*nd*	*nd*	*nd*	*d*	*d*
loaded	*nld*	*nld*	*ld*	*ld*	*ld*	*ld*
actions		*s*	*l*	*w*	*s*	*w*
caused E^+					*pd*	
caused E^-	*npd*	*npd*	*npd*	*npd*		

At step 4 the following is provided to the ILP system Progol.

$$
\begin{array}{ll}
B & nld(0).\ nld(1).\ ld(2).\ ld(3).\ ld(4).\ ld(5).\ \ldots \\
 & s(1).\ l(2).\ w(3).\ s(4).\ w(5).\ \ldots \\
 & d(4).\ d(5).\ \ldots \\
 & nd(0).\ nd(1).\ nd(2).\ nd(3).\ \ldots \\
E^+ & pd(4).\ \ldots \\
E^- & :\!-\ pd(0).\ :\!-\ pd(1).\ :\!-\ pd(2).\ :\!-\ pd(3).\ \ldots \\
\hline
H & pd(S) :\!-\ s(S),\ prev(S, PS),\ ld(PS).
\end{array}
$$

A single rule in H is enough to cover all the causality of dead. The solution to induction is H and the rules (step 5):

$$
\begin{array}{l}
d(S) :\!-\ prev(S, PS),\ d(PS),\ not\ pnd(S). \\
d(S) :\!-\ pd(S)
\end{array}
$$

Unfortunately the method is only applicable (see step 2) to narratives verifying a condition on the target fluent instances. In the next section the method will be extended to cope with this restriction.

Definition 2 (complete narrative) *A narrative is* target-complete *if there is a ground fact on the target fluent at every situation in the narrative.*

Theorem 1. *For narratives* target-complete *solutions by the method correspond one-to-one to solutions of induction of the effects of actions without the frame problem (Definition 1).* ■

Completeness and soundness of the method rely on the completeness (and soundness) of the monotonic ILP method used (It must be complete at least for solutions in the form of action laws and the additional restrictions required must also hold). The induction of the causality of the target fluent pf is induction in Horn LP, thus every solution is provided. On the other hand, the completeness and soundness also rely on the solution to the frame problem used being an actual solution to it, e.g., every solution with the frame problem has a corresponding representation with inertia axioms and vice versa. To complete the proof note that the examples on the causality of the target are selected to allow every possible causality: when there is a change in the fluent, it must be caused thus an action law is needed to cover these examples; but for negative examples only those on which the complementary fluent holds are provided, in these examples the fluent cannot be caused and true. For any other case, there can be solutions with or without the fluent caused. Finally, every solution by the method does not have the frame problem because no induction on persistent fluent instances is done.

5 Dealing with Missing Examples

Consider the following motivating example on a narrative in the YSS. There is a missing example on the target fluent at situation 3 (the narrative is not target-complete).

situation	0	1	2	3	4
dead	*nd*	*nd*	*nd*	?	*d*
loaded	*nld*	*nld*	*nld*	*ld*	*ld*
actions		*s*	*w*	*l*	*s*
caused E^+				*pd*?	*pd*?
caused E^-	*npd*	*npd*	*npd*	*npd*?	

By step 2 of the method causality is defined from two consecutive example instances. Then because of missing example at situation 3, no $pf(3)$ nor $pf(4)$ is obtained (nor $npd(3)$). Nevertheless it is clear that there is a change between situations 2 and 4. If a causality example is not provided here, target instance $d(4)$ will not be covered by some solutions, what makes the method unsound.

Missing example instances in the target predicate are common in learning. It is precisely because of these missing instances that alternative solutions to induction provide different generalizations; in this sense, more missing instances allow more generalization in the solution. Induction in action seems to behave the other way around, missing examples instead of facilitating induction turn it more complex. Note also that there is no problem with missing examples when induction is directly applied as showed before. But recall also that all these solutions have the frame problem. This points out the close relationship between dealing with missing examples and solving the frame problem in induction. In the discussion section, we comment on this.

The extension of the method is based on the following. Consider a narrative with a *missing segment*, i.e., several consecutive situations without example on the target fluent. Consider also that the examples just before and after the missing segment are complementary, e.g. nf and f; it is clear that there is a change in a situation in the missing segment or in the situation just after the segment, but the narrative does not tell where it is. Thus a missing segment like this represents the following input for induction: there is a positive example on causality among the situation instances inside the segment and the inmediate next situation.

Learning problems from examples provided as before have been already considered in machine learning under Multiple Instance (MI) learning. MI methods are able to deal with induction problems where each positive example is specified by a set of instances, instead of just one, meaning that at least one of the instances in the set is a positive example, but it is not known which one. MI methods can be represented in general ILP systems (see for instance [Finn *et al.*1998]) and we will follow this approach to extend the method for missing segments while still relying on regular ILP. The extension is presented by describing the additional tasks at the steps of the restricted method.

Step 1. (cont.) Define an extra argument for every action, e.g. $a(ES, S)$. For every missing segment in every narrative define a new constant to name it, e.g. $m34$, and give this constant as argument ES to every ground action inside the segment and to the action at the immediate situation after the missing segment, e.g. $l(m34, 3)$. For the rest of the action instances the ES argument is that of the situation, e.g. $w(2, 2)$. The extra argument on actions is used to denote the missing segment the action belongs to, in case there is one.

Step 2. (cont.) For every missing segment with complementary target instances at the situations immediately before and after the segment, a caused instance is defined as follows: If the situation inmediately after the segment has an f (resp. nf) instance define $pf(es)$ (resp. $pnf(es)$), where es is the constant name of the missing segment. Note that a single instance of causality is extracted from a missing segment, and that the instance is at the segment (es) as a whole.

Step 3. (no extension needed)

Step 4. (cont.) Now the induced action laws will have one of the forms:

$$pf(S) \text{ :- } a(S, S),\ prev(S, PS),\ f'(PS), \ldots \tag{14}$$

$$pf(ES) \text{ :- } a(ES, S),\ prev(S, PS),\ f'(PS), \ldots \tag{15}$$

Modify rules (15) by making the situation variable of pf that of S, the (regular) situation of the action in the rule. Then discard in every rule the extra situation argument of the action, so the rules become the usual action laws.

Step 5. (no extension needed)

For the previous example on the YSS with a missing segment, after the extensions at step 1 and 2 are applied we have,

situation	0	1	2	3	4
dead	*nd*	*nd*	*nd*	?	*d*
loaded	*nld*	*nld*	*nld*	*ld*	*ld*
actions		$s(1,1)$	$w(2,2)$	$l(m34,3)$	$s(m34,4)$
caused E^+					$pd(m34)$
caused E^-	*npd*	*npd*	*npd*		

After steps 3 and 4, the single rule induced H is

$$pd(ES) \text{ :- } s(ES, S),\ prev(S, PS),\ ld(PS).$$

that is transformed in the familiar rule $pd(S)$:- $s(S),\ prev(S, PS),\ ld(PS)$. And step 5 (as before) completes the solution.

Theorem 2. *Solutions by the extended method correspond one-to-one to solutions of induction of the effects of actions without the frame problem (Definition 1).* ■

Intuitively, the method works as follows. When induction has to cover one caused example at a missing segment $pf(es)$, any of the actions at the segment

can be choosen by using the reference es on them. Once a particular action is selected $a(es, s)$ the exact situation of change in the segment is also fixed, s, then any other reference for preconditions on other fluents is restricted to that situation s.

6 Discussion and Related Work

The method is applicable to induction under other causal formalisms of action [Lin 1995,Gustafsson and Doherty 1996] as the differences with the causality used here are not important for action laws.

Furthermore though causality is used during induction, it can be discarded in the final solutions making the method valid for action formalisms without causality. As an example the following transformation provides descriptions of actions in Answer Set Programming (ASP) [Lifschitz 1999].

Step 5. (alt.) (adapted for ASP) For every induced causal action laws (11) put the head directly on the original target $f(S)$, instead of the causal fluent pf.

Complete the solution adding the following inertia axiom for the original target $f(S)$, (Also for the complementary $nf(S)$, not shown.)

$$f(S) \text{ :- } prev(S, PS),\ f(PS),\ not\ nf(S). \tag{16}$$

This example points out that the descriptions induced are valid also for temporal explanation problems and for planning, besides temporal prediction. For instance, in ASP these problems can be solved after the addition of general rules for the fluents at the initial situation (temporal explanation), and general rules providing every possible sequence of actions (planning). These 'generation' rules do not depend on the behavior of the domain but on its signature (like inertia axioms).

One important area of application of the method is planning. The domain description in planning is usually based on STRIPS. The main difference with our causal formalism is the existence in STRIPS of a global unique precondition for every effect of an action. The method can be easily adapted to induce for these languages. On the other hand note that the solution to the frame problem in STRIPS-like representations is implicit, i.e. there is no explicit inertia axiom rule in the descriptions. Our method solves the frame problem in a general form that is also valid for STRIPS-like representations.

Two other approaches followed the same initial idea, namely induce action descriptions with ILP. In [Moyle and Muggleton 1997] the approach was introduced for Event Calculus descriptions, being [Moyle 2002] the most recent work. The methods proposed there are different and rely on working with NAF rules (for inertia) during induction, but it has been shown [Sakama 2000] [Otero 2001] that monotonic induction does not extend well to normal programs. The approach in [Lorenzo and Otero 2000] also uses some causality, in the sense that the examples on causality must be directly provided, and must be complete, i.e. a CWA on them is assumed, thus the so called nonmonotonic setting in induction is used. This restricts the range of applicability of the method, as causality

is usually not directly observable in the domains. In our approach causality is extracted from observations in the truth of fluents, and no CWA is assumed on it. Furthermore we allow also direct examples on the fluent pf. An important difference with these two approaches is that they are restricted to target-complete narratives. This can be understood as 'solving' the frame problem in induction by translating the frame problem to the evidence (set of examples) that must be complete, including every change and every non-change. As we mentioned before, non-monotonic ILP methods [Sakama 2000,Otero 2001] would provide alternatives to solve the frame problem. But currently they are defined under strong restrictions [Sakama 2000] and seem not enough to deal with the frame problem. Or they are not as efficient as monotonic ones [Otero 2001]. Note that even if a general nonmonotonic ILP method were available, methods using them would be less efficient than the one presented here.

References

[Finn *et al.*1998] P. Finn, S. Muggleton, D. Page, and A. Srinivasan. Pharmacophore discovery using the inductive logic programming system progol. *Machine Learning*, 30:241–271, 1998.

[Gustafsson and Doherty 1996] J. Gustafsson and P. Doherty. Embracing occlusion in specifying the indirect effects of actions. In *Proc. of the 5th Int. Conf. on Principles of Knowledge Representation and Reasoning*, 1996.

[Lifschitz 1999] V. Lifschitz. Action languages, answer sets and planning. In *The Logic Programming Paradigm: a 25-Year Perspective*, pages 357–373. Springer Verlag, 1999.

[Lin 1995] Fangzhen Lin. Embracing causality in specifying the indirect effects of actions. In *Proc. of the 14th International Joint Conference on Artificial Intelligence, IJCAI'95*, pages 1985–1991, 1995.

[Lorenzo and Otero 2000] D. Lorenzo and R. Otero. Learning to reason about actions. In *Proc. of the 14th European Conference on Artificial Intelligence, ECAI 00*, pages 316–320, 2000.

[Moyle and Muggleton 1997] S. Moyle and S. Muggleton. Learning programs in the event calculus. In *Proc. of the 7th Int. Workshop on Inductive Logic Programming, ILP 97, LNAI 1297*, pages 205–212, 1997.

[Moyle 2002] S. Moyle. Using theory completion to learn a robot navigation control program. In *Proc. of the 12th Int. Conf. on Inductive Logic Programming, ILP 02*, 2002.

[Muggleton 1995] S. Muggleton. Inverse entailment and Progol. *New Generation Computing*, 13:245–286, 1995.

[Muggleton 1998] S. Muggleton. Completing inverse entailment. In *Proc. of the 8th Int. Workshop on Inductive Logic Programming, ILP 98, LNAI 1446*, pages 245–249, 1998.

[Otero 2001] R. Otero. Induction of stable models. In *Proc. of the 11th Int. Conference on Inductive Logic Programming, ILP 01, LNAI 2157*, pages 193–205, 2001.

[Sakama 2000] C. Sakama. Inverse entailment in nonmonotonic logic programs. In *Proc. of the 10th Int. Conf. on Inductive Logic Programming, ILP 00, LNAI 1866*, pages 209–224, 2000.