Learning non-monotonic causal theories from narratives of actions

David Lorenzo
Computer Science Dept.
Univ. Coruña
15071 A Coruña (Spain)

Abstract

Non-monotonic formalisms for reasoning about actions and change have become a whole subfield of Artificial Intelligence. Current implementations allow to work with very expressive action theories involving ramifications, concurrent actions, complex qualifications and so on. A natural question that can be posed is whether this kind of declarative knowledge can be learned from observed time traces of property values from an existing dynamic system. For this task we consider a *narrative*-based logical theory of change in the form of Extended Logic Programs where logic-based learning methods can be applied effectively. The use of a narrative formalism provides more expressivity on the theories that are learned, for instance, to learn the effects of *concurrent* actions.

1 Introduction

Non-monotonic formalisms for reasoning about actions and change have become a whole subfield of Artificial Intelligence. These formalisms serve as an unifying approach that subsumes the many special purpose mechanisms that have been developed in other approaches, and that accommodates all the features of dynamical systems [24]. In these formalisms, system's behaviors are naturally viewed as appropriate logical consequences of the domain's description, hence the specification of actions and their effects is made intuitive and natural. Current implementations like VITAL [7], CCALC [17], PAL [4], among others, allow to work with very expressive action theories involving ramifications, concurrent actions, complex qualifications and so on.

A natural question that can be posed is whether this kind of declarative knowledge can be learned from observed time traces of property values from an existing dynamic system. Any attempt to learn such theories must cope with various facets such as inertia (and the associated Frame Problem), constraints and indirect effects (and the associated Ramification problem), non-deterministic and other complex effects of actions. Machine Learning methods have been used to learn planning operators [20, 5, 27], however, in most cases the inferred model corresponds to a set of STRIPS-like operators [8]. Some extensions have been considered since then, that are able to cope with more complex effects [3], however, they rely on special purpose formalisms, which makes the results of learning very dependent on the particular formalism and difficult to transfer.

Our work differs from previous approaches also in the use of Inductive Logic Programming [23] methods which allow a natural integration with implementations of Action Theories based on Logic Programming. Unlike classical inductive learning, ILP uses Logic Programming as the representational mechanism for hypotheses and observations. We adopt Extended Logic Programs as the form of programs to be learned, where two kinds of negation–negation as failure and classical negation–are effectively used in the presence of incomplete information.

This paper is organized as follows. Section 2 briefly describes logic-based formalisms for reasoning about actions and how these can be learned by adapting current *ILP* methods for static domains. In section 3 we consider the use of ramification constraints to learn the indirect effects of actions. In section 4, we describe a prototype based on conventional ILP methods and some experimental results. In section 5 we extend the framework to learn the effects of concurrent actions. Finally, in section 6 we present some conclusions and outlook future work.

2 Learning Action Theories from narratives

A fundamental problem in machine learning is the representation language used for the learning process. This is specially important when dealing with temporal data, where most of the previous methods treat temporal data as an unordered collection of events, ignoring its temporal information or need extensive transformations. In this paper, we consider the use of Action Languages [10] as the representational method underlying the learning process, which benefits from the declarative nature and the inferential capacity of these logic-based formalisms. The most classical formalisms for reasoning about actions and change are the Situation Calculus [18] and the Event Calculus [13]. In both cases, logic Programs can be used to represent the effects of actions by importing the corresponding ontology, where effect axioms are logic programs with a fixed clausal structure. In [16] we used an implementation of the Situation Calculus similar to [9], because of its simpler formalization with respect to the Event Calculus. However, Situation Calculus has several problems that limit its applicability [11], hence we have used a narrative-based version with a predicate *happens/2* to mean for the execution of actions, apart from the predicate *holds/2* that is used for recording the values of fluents at each situation ($s \geq 0$). This Logic Programming implementation uses an special predicate *caused/3* [14] (or *abnormal/3* [2]) and negation as failure to implement inertia. Formally we have that an action theory is a conjunction of:

- A finite set of general clauses $[\neg]Holds(f, 0)$ where 0 denotes the initial situation.
- A finite set of clauses in the form

$$Caused(f, v, t) \leftarrow Happens(a, t), \pi \quad (1)$$

  where π consists of literals in the form $[\neg]Holds(f', t\text{-}1)$. The description states that, in any situation, if the precondition holds then the effect will hold in the resulting situation. These axioms are called *effect axioms* or *action laws*.
- The universal frame axiom describes how the world stays the same (as opposed to how it changes).

$$Holds(f, t) \leftarrow Happens(a, t), Holds(f, t\text{-}1), \text{not } Caused(f, v, t) \quad (2)$$

$$\neg Holds(f, t) \leftarrow Happens(a, t), \neg Holds(f, t\text{-}1), \text{not } Caused(f, v, t) \quad (3)$$

- Some clauses that propagate caused values to $Holds$.

$$Holds(f, t) \leftarrow Caused(f, \text{true}, t) \quad (4)$$

$$\neg Holds(f, t) \leftarrow Caused(f, \text{false}, t) \quad (5)$$

Inertia is tackled in the following way: a fluent may only change if a cause for the change can be derived from the theory. Thus, we can add as many effect axioms as we like and rely upon axioms (2) and (3) to implement inertia. Other formalizations define a predicate *Pos/2* apart from *holds/2* and *caused/3*, such that for any action a and any situation s, $Poss(a, s)$ is true if a is executable in s. Thus, *caused/3* represents the *conditional effects* and *Poss* would represent the *executability* of the action. A similar shape for the programs to be learned based on the Event Calculus could also be used.

Given the description of an initial state of the world, and a set of observations of a dynamic system, the learning task we consider is to find a theory that correctly predicts the state of the world after the execution of any sequence of actions starting from any initial state. Let us consider the blocks world. Observations are in the form of *narratives* of actions (Fig. 1).

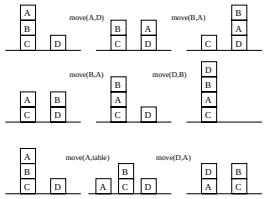

Figure 1: Narratives in the Blocks world

Each narrative consists of the values of fluents at the initial situation represented with the predicate *holds/2*, the actions executed and the effects produced by the actions, represented with the predicate *happens/2* and *caused/3* respectively. For instance, the first narrative of Fig. 1 is represented as follows:

$\text{holds}(clear(a), 0)$	$\text{happens}(move(a, d), 1)$	$\text{happens}(move(b, a), 2)$
$\text{holds}(clear(d), 0)$	$\text{caused}(on(a, b), f, 1)$	$\text{caused}(on(b, c), f, 2)$
$\text{holds}(on(a, b), 0)$	$\text{caused}(on(a, d), t, 1)$	$\text{caused}(on(b, a), t, 2)$
$\text{holds}(on(b, c), 0)$	$\text{caused}(clear(b), t, 1)$	$\text{caused}(clear(c), t, 2)$
$\text{holds}(on(c, table), 0)$	$\text{caused}(clear(d), f, 1)$	$\text{caused}(clear(a), f, 2)$
...		

so that *caused/3* represents the truth value and the causality that were represented separately by *holds/2* and *affects/3* in [16]. Each narrative may correspond to a different scenario or a different initial situation, which makes that examples covering a wide range of situations can be used for learning. In the Situation Calculus, multiple narratives of events are represented in a single model, however, they are now considered for learning as separate models, one for each narrative. This fits in the "learning from interpretations" paradigm of ILP where examples consist of multiple interpretations and not just one interpretation as it is usual in ILP. In our case, an interpretation corresponds to a narrative.

These observations can be provided directly to a ILP algorithm, however, the learner needs a full description

of each narrative, hence if a fluent does not change after executing an action, e.g., $holds(on(b,c),1)$, its truth value must be explicitly asserted in the input data. In previous approaches, both kinds of observations had to be provided as ground facts so that non-affected values of fluents must be part of the information given to the learning algorithm, thus having to provide the learner with large datasets for each narrative from which only a small fraction corresponds to effects of actions. In some cases, explicit *frame axioms* are obtained as part of the learning results [25]. This can be avoided if we include axioms (2 and 3) in the input data, so that observations need only be explicitly given for the effects of actions, whereas the inertia axiom propagates non-affected truth values from one situation to the next one, completing every situation. This is possible because of the explicit separation between the predicates *caused/3* and *holds/2*.

With respect to negative examples for each fluent, it is assumed that an action is not successful in the situations where the fluent did not change. For this reason, we build negative examples for the predicate *caused/3* for those fluents not affected by actions. By doing so, applying the induced rules on the observations will not result in postulating new effects not considered initially in the narratives used for learning. This CWA makes an inductive leap of a different kind, in the sense that it postulates that the induced rules will be valid on unseen narratives. However, if narratives are partially specified, additional methods are needed for learning to succeed [15].

Once narratives are encoded as shown above, conventional ILP methods can be applied to learn a definition for the positive and negative values of fluents. Positive examples E^+ consist of ground facts $caused(f,v,t)$ that correspond to the effects of actions, both for the positive value and for the negative value. Negative examples E^- consist of ground facts $caused(f,v,t)$ representing observations where a fluent f was not affected by any action. Apart from positive and negative examples, ILP uses so-called *Background Knowledge* (B), consisting basically of predicate definitions, and builds a definition of a target predicate in terms of itself and the background predicates. In our case, background knowledge includes a set of ground facts $happens(a,t)$ for every situation in a narrative, representing that an action a was executed at time t, *holds/2* ground facts for fluents at the initial situation of each narrative, the universal *inertia axiom* (eq. 2,3), axioms (4 and 5), and domain-dependent static predicates common to all narratives. The objective is to find a theory H composed of effect axioms in the form (1), such that:

$$(\forall e^+ \in E^+) \quad B \cup H \models e^+ \tag{6}$$

$$(\forall e^- \in E^-) \quad B \cup H \not\models e^- \tag{7}$$

In order to satisfy the completeness requirement, the learned rules for the positive and the negative concept will entail all positive examples (eq. 6). The consistency requirement is satisfied when the induced rules do not entail any of the negative examples (eq. 7).

3 Learning ramifications of actions

Previous approaches to learning action models are restricted to predicting a single outcome or effect of an action. This forces the explicit representation of all the effects of an action as direct effects, producing the so-called *Ramification problem*. In some cases, it is difficult to express some action preconditions explicitly as a condition on the starting state. This makes effect axioms increasingly complex and consequently harder to learn, since they need to anticipate the *ramifications* of the executed action. Most solutions that have been proposed for the ramification problem require effect axioms to specify the most significant effects of an action and rely on *ramification constraints*, i.e., general laws describing dependences between fluents, for specifying additional changes that are due to the action. A ramification constraint is a formula

$$caused(f,v,t) \leftarrow \pi \tag{8}$$

where the $Holds$ literals in π are only of the form $[\neg]Holds(f',t)$. According to this, indirect effects are derived from the state of fluents, while direct effects come from the execution of actions.

The learner must now infer how properties of a domain are (directly/indirectly) affected by the execution of actions. In general, actions can have multiple direct and indirect effects, and fluents can be a direct (resp. indirect) effect of multiple actions. In practice, we allow the learner to determine at each step whether a fluent should be learned as a direct or an indirect effect, thus learning possibly a mix of axioms based on a compression-based measure so that smaller theories are preferred. The insight is that indirect effects can make programs sensibly shorter, given that a ramification constraint may subsume the direct effects of several actions, which have a positive influence in their learnability, as the difficulty of learning a logic program is very much related to its length.

Indirect effects actually represent a *propagation of changes*, hence ramification constraints are allowed to cover only those examples where the action executed produced several simultaneous effects, and at least one of the fluents in the body must have changed simultaneously with the fluent in the head. Note also that the predicate *caused/3* cannot be substituted by

holds/2 in (8) because a fluent that has been initiated/terminated directly through an effect axiom cannot be terminated/initiated indirectly through such a ramification constraint, unless it is released from inertia beforehand, otherwise it would lead to contradiction. However, a number of "useless" (even if correct) instances of *caused/3* are derived even for those situations where a fluent does not change. This contradicts the observations given that negative examples for a fluent are given for those situations where the fluent does not change. To avoid this, the consistency of ramification constraints is not tested on those negative examples where a fluent does not change its value.

4 Learning with conventional ILP methods

We have developed a prototype *LRAC* consisting of an "ordinary" ILP algorithm. We do not assume any particular learning algorithm, however, in the prototype we used the notion of Inverting Entailment (IE) that is the technique used in the system Progol [22]. The IE method computes the most specific generalization (also called bottom clause) from a positive example with respect to the background knowledge. For instance, the most specific generalization of $caused(on(c, table), t, 4)$ is[1]:

```
caused(on(A,B),t,C) :- happens(move(A,B),C), diff(A,B), diff(B,A),
        -holds(on(A,B),C-1), -holds(on(B,A),C-1), holds(on(A,E),C-1),
        holds(on(B,F),C-1), holds(clear(A),C-1), holds(clear(B),C-1),
        -holds(on(A,F),C-1), -holds(on(B,E),C-1), -holds(on(F,A),C-1),
        -holds(on(F,B),C-1), -holds(clear(F),C-1), holds(on(F,E),C-1),
        diff(A,E), diff(A,F), diff(B,E),diff(B,F), diff(E,A), diff(E,B),
        diff(E,F), diff(F,A),diff(F,B), diff(F,E), table(E).
```

Effect axioms introduce a bias for the clauses to be learned, where *holds/2* literals in the body can only refer to the previous situation. In general, the bottom clause can have infinite cardinality, however some biases are considered to keep it manageable, e.g., type information for fluent and action literals, bounds on the length of hypotheses and the depth of variables, and other syntactic restrictions. *LRAC* searches in a top-down fashion to find a subset of literals of the bottom clause that explains as many of the positive examples as possible, so that the search space is bounded by this most specific clause and the most general clause (with an empty body). Then the explained examples are separated and the algorithm recursively conquers the remaining positive examples.

The rule generation phase will be called twice for each fluent, once for the positive concept and once for the negative concept, and will be repeated searching for effect axioms and for ramification constraints. For testing the coverage of hypotheses, these are evaluated by performing a derivation of each example from a program composed by the hypothesis, the background theory and the clauses previously learned, so that the coverage test corresponds to a *temporal projection* problem for each hypothesis generated by *LRAC*, which is a high overload for medium-sized domains. However, the common approach in ILP is to use θ-subsumption which is computationally more efficient so that a clause covers *extensionally* an example e, i.e., there exists a ground instance of the clause $e \leftarrow l_1, \ldots, l_n$ where each l_i belongs to $B \cup \{e\}$. However, θ-subsumption needs some assumptions to guarantee completeness and soundness [15]. For instance, care must be taken to ensure that the addition of ramification constraints to a theory does not cause a *non-finite recursion*, where fluents may use one another in their definitions. If a cycle is identified, the last clause is removed and remembered, so that in the case in which it is generated again in the specialization loop, it is immediately discarded. This is done in order to avoid that the system goes into a loop of continuously generating and retracting the same clause.

For the experimental part we have considered several challenge problems in the literature of reasoning about actions, as well as other domains used in AIPS planning competitions [19]. The objective is to show not only if a sufficiently general theory can be learned from observations but also if the learned theory for each domain matches the description found in the literature. Let us consider a prototypical problem of the literature [26], that consists of a circuit that includes a lamp, a relay, and three switches sw_1, sw_2 and sw_3, together with actions in the form t_i $(i = 1 \ldots 3)$.

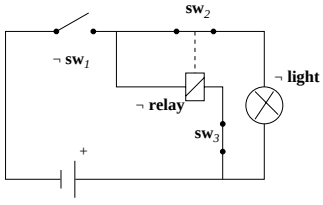

Figure 2: An electric circuit

We used a simple *wander* program that collects data about its actions and the fluents that changed while exploring the environment. Input data consists of 21 narratives from which *LRAC* found:

```
caused(sw1,t,A) :- happens(t1,A),-holds(sw1,A-1).
caused(sw1,f,A) :- happens(t1,A),holds(sw1,A-1).
caused(sw2,t,A) :- happens(t2,A),-holds(sw2,A-1),
                   -holds(relay,A-1).
caused(sw2,f,A) :- happens(t2,A),holds(sw2,A-1).
```

[1] Background for the blocks world consists of domain predicates $on/2$, $clear/1$, $block/1$, $table/1$, and a predicate $diff/2$ to represent $X \neq Y$.

```
caused(sw2,f,A) :- holds(relay,A).
caused(sw3,f,A) :- happens(t3,A),holds(sw3,A).
caused(sw3,t,A) :- happens(t3,A),-holds(sw3,A).
caused(light,t,A) :- holds(sw1,A), holds(sw2,A).
caused(light,f,A) :- -holds(sw1,A).
caused(light,f,A) :- -holds(sw2,A).
caused(relay,t,A) :- holds(sw1,A), holds(sw3,A).
caused(relay,f,A) :- -holds(sw1,A).
caused(relay,f,A) :- -holds(sw3,A).
```

According to the rules, action t_1 (resp. t_3) toggles switch sw_1 (resp. sw_3). The *relay* is controlled by switches sw_1 and sw_3, i.e., the relay is active whenever both sw_1 and sw_3 hold simultaneously, so that only ramification constraints are learned for the relay instead of the corresponding effect axioms for all the actions that affect it, i.e., t_1 and t_3, thus producing a shorter theory. Similarly for *light* with sw_1 and sw_2. With respect to sw_2, the negative value is in some cases a direct effect of t_2 so that sw_2 can be closed only when the relay was previously inactive, and in other cases an indirect effect of the actions that activate the relay. The use of the ramification constraint for the negative value also produces a shorter theory, otherwise *LRAC* would need to learn an effect axiom for the actions that activate the relay.

In the blocks world, from 48 narratives of length 6, *LRAC* learned the following clauses:

```
caused(on(A,B),t,C) :- happens(move(A,B),C),
                       holds(clear(A),C-1),holds(clear(B),C-1).
caused(on(A,B),t,C) :- happens(move(A,B),C),
                       holds(clear(A),C-1),table(B).
caused(on(A,B),f,C) :- holds(on(A,D),C),diff(B,D).
caused(clear(A),t,B) :- happens(move(C,D),B),
                        holds(clear(D),B-1),holds(clear(C),B-1),
                        holds(on(C,A),B-1),diff(D,A).
caused(clear(A),t,B) :- happens(move(C,D),B),holds(clear(C),B-1),
                         holds(on(C,A),B-1),table(D).
caused(clear(B),f,A) :- holds(on(_,B),A).
```

LRAC learned two effect axioms for the positive value of $on/2$ that deal separately with the cases where a block is moved onto other block or onto the table [2]. The use of ramification constraints produces a shorter theory for the negative value of $on/2$, even with a single action $move/2$, and it represents that $on/2$ is a *functional fluent*. The ramification constraint for the negative value of $clear/1$ does not produce a clear benefit mainly because there are no other actions that affect it apart from $move/2$. However, as the ramification constraint is shorter than the corresponding effect axiom, *LRAC* is biased to prefer it. Actually the positive value of $clear/2$ is also an indirect effect, given that a block is clear if there is no another block on it, however, *LRAC* learned two effect axioms for the $move/2$ action due to the limited quantification of logic programs.

[2] The table is an special location that is always clear.

5 Concurrent actions

We have also assumed that actions are atomic. If *concurrent actions* are allowed, then an effect may depend on a particular combination of actions, an action may qualify another action's effects, effects can be cancelled, and so on [1]. As a consequence, learning about concurrent actions even in simple domains, is a complex task because the theory to be learned must cope with every possible interaction between actions. In this case, negative examples correspond to those situations where an effect is not caused because there are missing preconditions, missing actions or actions that are executed additionally.

Concurrent actions are introduced by allowing multiple instances of the predicate *happens/2* in the body of rules to be learned. Default negation is used to cope with the non-execution of conflicting actions. We explicitly use positive and negative *happens/2* literals so that at least a positive *happens/2* literal must be added to the body, whereas the negative counterpart of *happens/2* will include any potentially cancelling action, which are taken from those actions executed in the negative examples. By doing so, *LRAC* can specialize an overgeneral effect axiom by adding positive and negative *holds/2* and *happens/2* literals. Effects of concurrent actions can be also seen in terms of inheritance [1], i.e., compound actions normally inherit effects from their subactions, and cancelling actions cancel inheritance of the effects of atomic actions. However, we use positive and negative *happens/2* literals instead of cancelling and inheritance axioms, given that the latter express separately the positive occurrences of actions and their preconditions from the cancelling actions and their preconditions, and it is not obvious how they can be learned separately.

Let us consider again the circuit of Figure 2. Fluents sw_1 and sw_3 are independent effects, hence the effect axioms learned for them do not differ from those learned when a single action is executed. With respect to the state of *light*, the consequences of executing, e.g. t_1, depend on the previous value of sw_2 in a different way depending on whether t_2 is also executed or not simultaneously. For instance, when both switches are closed simultaneously, they produce independent effects and the accumulative effect of activating the light. As a consequence, many more clauses have to be learned to cope with every possible interaction between actions. However, *LRAC* learned the same definition for *light*, because the ramification constraints express a dependence between the switches and *light* without making reference to the actions executed. Similarly for the *relay* that is controlled by

sw_1 and sw_3. With respect to sw_2, $LRAC$ found the following definition:

```
caused(sw2,f,A) :- happens(t2,A),holds(sw2,A-1).
caused(sw2,f,A) :- holds(relay,A).
caused(sw2,t,A) :- happens(t2,A),
                   not happens(t1,A),not happens(t3,A),
                   -holds(sw2,A-1),-holds(relay,A-1).
caused(sw2,t,A) :- happens(t2,A),not happens(t1,A),
                   -holds(sw2,A-1),-holds(sw1,A-1).
caused(sw2,t,A) :- happens(t2,A),not happens(t3,A),
                   -holds(sw2,A-1),-holds(sw3,A-1).
caused(sw2,t,A) :- happens(t2,A),happens(t1,A),happens(t3,A),
                   -holds(sw2,A-1),-holds(relay,A-1),
                   holds(sw1,A-1).
caused(sw2,t,A) :- happens(t2,A),happens(t1,A),happens(t3,A),
                   -holds(sw2,A-1),-holds(relay,A-1),
                   holds(sw3,A-1).
```

When only atomic actions are executed, $LRAC$ learned two effect axioms and one ramification constraint. However, if we execute t_2 and other action that activates the *relay* at the same time (e.g., t_3), sw_2 may be closed for an instant but it will be definitely open when the relay becomes active, so that the effect axiom for the positive value of sw_2 needs to consider all subsets of cancelling actions that could activate the relay, together with their preconditions. This makes the descriptions of actions cumbersome and difficult for complex domains, and consequently harder to learn.

In most current action formalisms (VITAL [7], CCALC [17], etc.), such complex *qualifications* to actions are often expressed apart from the effect axioms in the form of *integrity constraints* that forbid some next states produced by effect axioms, that correspond to non-allowed states of a domain [12]. For instance, in the previous circuit, we can extract a set of constraints that correspond to those states not allowed for the switches, for instance, that sw_2 is never closed when the relay is active. From such constraint, we have the following alternative definition for the positive value of sw_2:

```
caused(sw2,t,A) :- happens(t2,A),-holds(sw2,A-1).
:- holds(sw2,A),holds(relay,A).
```

In this case, the constraint specializes the overgeneral effect axiom for sw_2, so that only valid and legal resulting states are allowed. By doing so, the state of sw_2 does not depend on what other actions are executed apart from t_2, but on the state of the *relay* in the resulting situation, which provides with a much simpler definition. A consequence is that these constraints can make programs sensibly shorter and thus easier to learn. The interest of such qualification constraints is clearer when learning the effects of concurrent actions, since we may need to anticipate the effects of the co-occurring actions. We are currently considering to learn such qualification constraints to specialize overgeneral effect axioms instead of fully specialize them by adding preconditions. The choice of the constraints is not independent from the rules of the theory, and the difficulty lies in finding the "relevant" constraints that would compensate correctly for the rules of the theory [6].

6 Conclusions

A logic programming formalization of action domains has been well-studied, yet not much previous work [21] exists on the combination with learning methods. The presented approach improves on previous ones by providing more expressivity, so that it can solve some problems that were unsolvable before. Further work needs to be done to show the adaptation to more and increasingly more complex scenarios, and with different noise levels, e.g., to improve its adequation for dealing with real robot's environments. More complex phenomena can be dealt with in an homogeneous way, by slightly changing the form of the rules to be learned or building special fluents, e.g., for durative actions or delayed effects. Further work also includes the management of continuously varying parameters as a consequence of a process execution.

Acknowledgements

This research was supported in part by the Government of Spain, grant TIC2001-0393.

References

[1] C. Baral and M. Gelfond. Reasoning about effects of concurrent actions. *Journal of Logic Programming*, 31(1–3):85–117, 1997.

[2] C. Baral and J. Lobo. Defeasible specifications in action theories. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI-97)*, pages 1441–1446, San Francisco, 1997. Morgan Kaufmann Publishers.

[3] S. Benson. Inductive learning of reactive action models. In *Proc. 12th International Conference on Machine Learning*, pages 47–54. Morgan Kaufmann, 1995.

[4] P. Cabalar, M. Cabarcos, and R. P. Otero. PAL: Pertinence action language. In *Proceedings of the 8th Intl. Workshop on Non-Monotonic Reasoning NMR'2000 (Collocated with KR'2000)*, Breckenridge, Colorado, USA, 2000.

[5] J. G. Carbonell and Y. Gil. Learning by experimentation: The operator refinement method. In

R. S. Michalski and Y. Kodratoff, editors, *Machine Learning: An Artificial Intelligence Approach, Volume III*. Morgan Kaufmann, San Mateo, California, 1990.

[6] Y. Dimopoulos, S. Džeroski, and A. Kakas. Integrating explanatory and descriptive learning in ILP. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI-97)*, pages 900–907, San Francisco, 1997. Morgan Kaufmann Publishers.

[7] P. Doherty and J. Kvarnstrom. Tackling the qualification problem using fluent dependency constraints: Preliminary report. In *International Workshop on Temporal Representation and Reasoning*, pages 97–104, 1998.

[8] R. E. Fikes and N. J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. In D. C. Cooper, editor, *Proceedings of the 2nd International Joint Conference on Artificial Intelligence*, pages 608–620, London, UK, 1971. William Kaufmann.

[9] M. Gelfond and V. Lifschitz. Representing action and change by logic programs. *Journal of Logic Programming*, 17:301–321, 1993.

[10] M. Gelfond and V. Lifschitz. Action languages. *Electronic Transactions on AI*, 3(16), 1998.

[11] M. Gelfond, V. Lifschitz, and A. Rabinov. What are the limitations of the situation calculus? In *Automated Reasoning: Essays in Honor of Woody Bledsoe*, pages 167–179. Kluwer Academic Publishers, Dordrecht, 1986.

[12] Matthew L. Ginsberg and David E. Smith. Reasoning about action II: the qualification problem. In Frank M. Brown, editor, *The frame problem in artificial intelligence: proc. of 1987 workshop*. Morgan Kaufmann, 1987.

[13] R. Kowalski and M. Sergot. A logic-based calculus of events. *New Generation Computing*, 4:67–95, 1986.

[14] F. Lin. Embracing causality in specifying the indirect effects of actions. In C. S. Mellish, editor, *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*. Morgan Kaufmann, 1995.

[15] D. Lorenzo. *Learning non-monotonic Logic Programs to Reason about Actions and Change*. PhD thesis, Departamento de Computación, Facultade de Informática, Univ. A Coruña, Spain, 2001.

[16] D. Lorenzo and R.P. Otero. Learning to reason about actions. In *W. Horn (Ed.), Proceedings of the 14th European Conference on Artificial Intelligence*, pages 435–439. IOS Press, Amsterdam, 2000.

[17] N. McCain and H. Turner. A causal theory of ramifications and qualifications. In *Proc. of the Intl. Joint Conf. on Artificial Intelligence (IJCAI)*, pages 1978–1984, 1995.

[18] J. McCarthy and P.J. Hayes. Some philosophical problems from the standpoint of artificial intelligence. *Machine Intelligence*, 4:463–502, 1969.

[19] D. McDermott. The 1998 ai planning systems competition. *AI Magazine*, 21(2):35–55, 2000.

[20] T. M. Mitchell, P. E. Utgoff, and R. Banerji. Learning by experimentation: acquiring and refining problem-solving heuristics. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach*, pages 163–190. Morgan Kaufmann, San Mateo, California, 1983.

[21] S. Moyle and S. Muggleton. Learning programs in the event calculus. In N. Lavrač and S. Džeroski, editors, *Proceedings of the Seventh Inductive Logic Programming Workshop (ILP97)*, LNAI 1297, pages 205–212, Berlin, 1997. Springer-Verlag.

[22] S. Muggleton. Inverse entailment and Progol. *New Generation Computing, Special issue on Inductive Logic Programming*, 13(3-4):245–286, 1995.

[23] S. Muggleton and L. De Raedt. Inductive logic programming: Theory and methods. *Journal of Logic Programming*, 19/20:629–679, 1994.

[24] R. Reiter. *Knowledge in action: Logical Foundations for describing and implementing dynamical systems*. MIT Press, 1998.

[25] W. M. Shen. *Autonomous Learning from the Environment*. Computer Science Press, 1994.

[26] M. Thielscher. Ramification and causality. *Artificial Intelligence Journal*, 1-2(89):317–364, 1997.

[27] X. Wang. Learning by observation and practice: an incremental approach for planning operator acquisition. In *Proc. 12th International Conference on Machine Learning*, pages 549–557. Morgan Kaufmann, 1995.