Learning to Reason about Actions

David Lorenzo[1] and **Ramon P. Otero**[2]

Abstract. We focus on learning representations of dynamical systems that can be characterized by logic-based formalisms for reasoning about actions and change, where system's behaviors are naturally viewed as appropriate logical consequences of the domain's description. To this end, logic-based induction methods are adapted to identify the input/output behavior of a dynamical system corresponding to an environment. The study of dynamic domains is started with domains modelable with classical action theories and is progressively enhanced to manage more complex behaviors.

1 Introduction

Dynamical system identification is defined as inference of the causal laws relating actions to their effects where inputs correspond to the actions executed and outputs correspond to the perceptual information available in particular states of an environment. System identification has been studied in a variety of disciplines including control theory, neural networks, and automata theory. The inferred model might correspond to a system of differential equations, a set of production rules [13], or a set of states and transition probabilities.

We focus on learning representations of environments that can be characterized by logic-based formalisms for *reasoning about actions and change*. These formalisms aim to be an unifying theory of dynamics, grounded on a mathematical and computational foundation, where system's behaviors are naturally viewed as appropriate logical consequences of the domain's description. To address this problem is a non trivial task, taking into account the range of phenomena to accommodate, among others:

- The causal laws relating actions to their effects.
- The conditions under which an action can be performed.
- Unreliable sensors.
- Environments partially known to an agent.

Recently there has been a lot of progress in formulating theories of actions, particularly in progressing from simple and/or restricted theories and 'example centered approaches', to general theories that incrementally consider various specification aspects. This allows that more complex problems can be managed, including *Cognitive Robotics* aimed at the construction of robots whose architecture is based on the idea of representing the world by sentences of formal logic and reasoning about it by manipulating those sentences [12, 1].

Most of these formalizations define an entailment relationship ($\models$) between the specifications (of effects of actions and relation among objects of the world). This ability allows to make plans that will take us to particular kind of worlds and explain observations about the state of the world. As to learning, the problem is somewhat the contrary. The apprentice is provided with a sequence of states that leads the system to a desired or undesired final state. Then, it must infer how properties of a domain are (directly/indirectly) affected by the execution of actions, or otherwise are subject to the general law of persistence, that helps explain the final state.

This task needs the use of knowledge during learning, as well as non-monotonic reasoning capabilities, where both default and classical negation are available, given that an apprentice needs to distinguish between what is true, what is false and what is unknown. The use of Logic Programming (LP) makes it feasible to study the integration of Inductive Logic Programming (ILP) with formalisms for reasoning about actions. Specific challenges for action theories, of which the most famous are, the Frame Problem, the Qualification Problem, and the Ramification Problem, are also present in learning.

This paper is organized as follows. Section 2 briefly describes formalisms for reasoning about actions. Sections 3 and 4 show how classical action theories can be learned by adapting current ILP methods for static domains, whereas in section 5 consistency requirement is relaxed to allow the induction of default action theories. The paper concludes with some future directions for research.

2 Action Theories

A formal framework to reason about actions and change requires the basic notion of a *situation*, which is a partial snapshot of the world at a particular instant of time. A *domain description* consists of two nonempty sets: a set $\mathcal{F}$ of fluent names, and a set $\mathcal{A}$ of action names. An action, if executed in some state, leads to a "resulting" state. So-called fluents serve to describe situations and are properties whose truth values may change in the course of time.

This work deals with a logical approach to modeling dynamical systems based on a dialect of first order logic called the Situation Calculus [8]. SC programs are logic programs with a fixed clausal structure where any aspect of a system which can change as the result of an action is indexed by a situation. A situation argument may be in the form of a constant s_i or else a situation resulting after executing an action $do(action, sit)$, and a special situation s_0 is included to represent the initial situation. A sequence of situations is encoded in the form of chains `do(act,do(act,..do(act,s)..))` starting from the initial situation.

Logic programming (LP) can be used to represent the effects of actions by importing the ontology of SC [11, 2], representing effect axioms as logic program clauses and using negation-as-failure (NAF) as a means of overcoming the frame problem (section 3). Formally we have

Definition 1 (Situation Calculus Program) A SC program is the conjunction of [11]:

[1] Computer Science Dept., University of Corunna, A Corunna, Spain, email: david@dc.fi.udc.es
[2] Computer Science Dept. Univ. Texas at El Paso, USA, email: otero@cs.utep.edu

- A finite set of general clauses
$$[\neg]Holds(f, s_0) \tag{1}$$
where s_0 denotes the initial situation.
- A finite set of clauses of the form
$$[\neg]Holds(f, do(a, s)) \leftarrow \pi \tag{2}$$
where π does not mention the *Affects* predicate and every occurrence of the Holds predicate in π is of the form $Holds(F', s)$. 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*.
- A finite set of *Affects* clauses of the form
$$Affects(a, f, s) \leftarrow \pi \tag{3}$$
where π does not mention the *Affects* predicate and every occurrence of the *Holds* predicate in π is of the form $Holds(f', s)$.
- The universal frame axiom describes how the world stays the same (as opposed to how it changes).

$$Holds(f, do(a, s)) \leftarrow Holds(f, s) \wedge \text{not } Affects(a, f, s) \tag{4}$$
$$\neg Holds(f, do(a, s)) \leftarrow \neg Holds(f, s) \wedge \text{not } Affects(a, f, s) \tag{5}$$

□

Most previous work on ILP consider definite Horn programs. However, research work on Reasoning about actions has shown that such monotonic programs are not adequate to represent the effects of actions [2]. To overcome this and some derived problems, the class of *extended logic programs* (ELP) [4] under the well-founded semantics is adopted as the language representation.

3 Induction of Action Theories

The basic problem is to identify the effects of actions, values of fluents (observations) and relationships between them, describing how they progress a domain where they are performed. Rather than directly learning a strategy for acting, the apprentice must learn a model of the world. Unlike [13] where the objective is to explore the domain by acting on it, we consider the learning process and the exploratory process as two separate processes, such that we are limited to a finite dataset.

Let us consider the following problem.

Example 1 (Circuit 1**)** *A simple circuit that includes a lamp, a relay, and three switches sw_1, sw_2 and sw_3, together with some actions in the form $toggle(sw_i)$.*

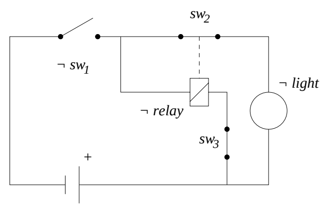

Figure 1. An electric circuit

□

This example concerns the representation of knowledge about the objects in a circuit, and how such knowledge is acquired. In this case, machine learning helps to construct the description of the circuit from actions, where measurements accumulated during simulations are the input to learning. We assume that actions are separated from their consequences, i.e., an agent can execute its actions in any environment regardless of which consequence the action might cause. Furthermore, actions may occur apart from those needed to explain the facts. A situation in which many irrelevant actions may be present is when the same training data is being used to learn many different concepts, as it is our case.

Input data consists of `holds/2` observations for the initial situation and for every situation included in the examples, representing the truth values for fluents at each situation.

```
-holds(sw1,s0).
holds(sw2,s0).
holds(sw3,s0).
-holds(light,s0).
-holds(relay,s0).
...
holds(sw1,do(t1,s0)).
-holds(sw2,do(t1,s0)).
holds(relay,do(t1,s0)).
  ...
```

Examples are then observations of the shape $holds(f, do(a, s))$ (resp. $\neg holds(f, do(a, s))$), whereas negative examples are negations of observations. To allow multiple sequences, we include different initial situations s_0^i, so that not all sequences must begin in the same state (same values for all fluents).

The use of logic-based formalisms raises specific challenges for action theories, of which the most famous is the Frame Problem, that is also present in learning. When we specify action laws we are only interested in describing changes, assuming that all the rest will remain unaffected. LP uses negation-as-failure (NAF) as a means of overcoming the frame problem through the Universal inertia axiom. However the frame problem is *also present in the evidence*, namely, if a fluent does not change after executing an action, its truth value must be explicitly asserted in the input data. A system without the inertia principle would have an intractable, or even impossible, representation. The inclusion of inertia values as examples for a fluent f, we should induce an inertia axiom (for completeness) for each pair (a, f) where action a does not change fluent f.

$$[\neg]Holds(f, do(a, s)) \leftarrow [\neg]Holds(f, s) \tag{6}$$

Furthermore, as the apprentice cannot distinguish caused values from inertia values, it will try to find a consistent clause that might cover both types of examples [6]. To avoid this, inertia values should be "separated" before learning and used as background knowledge (BK). With the inclusion of axioms (4,5) in the BK, positive examples are only explicitly given for those situations where a fluent changes, whereas the inertia axiom propagates not affected truth values from one situation to the next one, completing every situation. An important consequence is that induction is done only over *caused values*. This requires that positive `affects/3` atoms are generated for caused values of fluents and added to BK. In the final theory, rules for `affects/3` are duplicates of those for `holds/2`.

On the other hand, due to the inertia assumption, once we supply values for all fluents in the initial situation, we specify values for all possible situations and thus, situations are completed for those fluents that have an unknown value. As to evidence, we need a fluent to be *affected* to disable inertia, e.g., when it becomes unknown after executing an action. To achieve it, additional `affects/3` atoms should be used for missing values. As to the final theory, we need to disable the inertia rule not only when the preconditions for the change in the value of a fluent are known to hold, but whenever there is no evidence that they do not hold [2]. This requires that rules for predicate `affects/3` in the final theory are to be in the form:

$$Affects(a, f, s) \leftarrow \text{not } \neg Holds(f', s), \ldots . \tag{7}$$

We therefore consider the following learning problem, where E contains examples for m fluents such that E^+ and E^- are divided into m subsets $E^+_{f_i}$ and $E^-_{f_i}$.

Definition 2 (Learning SC Programs)
Given (for each $f_i \in \mathcal{F}$ and $a \in \mathcal{A}$):

- A set $E^+_{f_i}$ of positive examples (ground facts) $[\neg]holds(f_i, do(a, s))$, representing observations where f_i changed.
- A set $E^-_{f_i}$ of negative examples (ground facts).
- Background knowledge (BK), including `holds/2` ground facts for fluents at the initial situation s_0, `holds/2` ground facts for every $f_j \neq f_i$ and $s \neq s_0$, `affects/3` ground facts for every e such that $\exists k : e \in E^+_{f_k}$, and the universal inertia axiom (ax. 4,5).

Find a SC program H_{f_i}, such that:

$$(\forall e^+ \in E^+_{f_i}) \;\; BK \cup H \models e^+$$
$$(\forall e^- \in E^-_{f_i}) \;\; BK \cup H \not\models e^-$$

□

4 Implementation

A prototype consisting of a top-down ILP algorithm has been integrated with a Prolog interpreter and implemented in XSB Prolog. The design methodology is to present the user with a Prolog interpreter augmented with inductive capabilities where the syntax for examples, background knowledge and hypotheses is the Prolog syntax. This makes possible to include arbitrary Prolog programs as background knowledge by calling the Prolog interpreter to derive ground atoms from intensionally coded specifications of background predicates, and default negation.

The implemented ILP algorithm is mostly based on Progol [9]. The search is delimited by the empty clause and the so-called $\bot$-clause (most specific generalization) constructed from a *seed* and the BK, whose size and form are controlled by applying a syntactic bias. Search is done in a top-down fashion, adding literals until a consistent clause is found with maximal compression. The induction problem is decomposed into two separate problems, one for learning true values of fluents, and one for false values, and independently for each action that affects the fluent. The order of learning the fluents is determined by the seeds.

```
1. If $E = \varnothing$ return $H = \varnothing$
2. Let $e$ be the first example in $E$
3. Construct clause $\bot$ for $e$
4. Construct clause $H_e$ from $\bot$
5. Let $H = H \cup H_e$
6. Let $E' = \{e : e \in E$ and $H \cup BK \models e\}$
7. Let $E = E - E'$
8. Goto $1$
```

Figure 2. Progol covering algorithm

Two situations are included in a SC formulae, namely, the previous situation and the situation resulting of executing an action. We introduce an implicit *bias* for the clauses to be induced, where only the previous situation can appear in the body of a hypothesized clause. Thus, any literal added to a clause `holds(F,do(a,S)) :- ...` will refer to S (not to `do(a,S)`). This is the form of so-called *effect axioms*.

As a first step, positive observations are converted into LP syntax and negative examples and Background Knowledge are generated. Negative examples are given for every positive example `[-]holds(f,s)` when the opposite value holds (persists or is caused). For simplicity, noise is treated as a kind of non-determinism, and dealt with by relaxing the consistency requirement (see section 5), whereas for unknown values inertia is applied (by default) over the most recent situation where the fluent was known.

For instance, in example 1, we obtained the following clauses[3]:

```
| ?- domain(circuit1).
| ?- induce.
-holds(sw1,do(t1,A)) :- holds(sw1,A).
holds(sw1,do(t1,A)) :- -holds(sw1,A).
-holds(sw3,do(t3,A)) :- holds(sw3,A).
holds(sw3,do(t3,A)) :- -holds(sw3,A).
-holds(light,do(t1,A)) :- holds(light,A).
-holds(light,do(t2,A)) :- holds(light,A).
-holds(light,do(t3,A)) :- holds(light,A).
holds(light,do(t1,A)) :- -holds(sw1,A), holds(sw2,A).
holds(light,do(t2,A)) :- holds(sw1,A), -holds(sw2,A).
-holds(sw2,do(t1,A)) :- holds(sw2,A), holds(sw3,A).
-holds(sw2,do(t2,A)) :- holds(sw2,A).
-holds(sw2,do(t3,A)) :- holds(light,A).
holds(sw2,do(t2,A)) :- -holds(sw2,A), -holds(relay,A).
-holds(relay,do(t1,A)) :- holds(relay,A).
-holds(relay,do(t3,A)) :- holds(relay,A).
holds(relay,do(t1,A)) :- holds(sw3,A), -holds(relay,A).
holds(relay,do(t3,A)) :- holds(sw1,A), -holds(sw3,A).
```

According to the rules, action t_1 (resp. t_3) toggles switch sw_1 (resp. sw_3), such that no other action modifies them. For the rest of fluents, the system induced 5 clauses for $light$ and 4 for $relay$ –where all actions (t_1, t_2, t_3) affect them–, and 4 for sw_2.

The syntactic bias forces the explicit representation of all the effects of an action as direct effects, producing the so-called *ramification problem*. For instance, according to Fig. 1, the $light$ is on when both sw_1 and sw_2 are closed, and however they actually affect $light$ not directly but through actions that modify them (t_1, t_2, t_3). The result is that effect axioms need more clauses to cope with all possible cases, hence the inductor has to produce a high number of clauses based possibly on little evidence, thus the induced hypotheses may be unnecessarily complex and thereby also less reliable and accurate. The consequences increase as more dependences among fluents exist in the domain.

Actually, $light$ is an *indirect effect* of the switches. Such indirect effects are usually represented as consequences of general laws describing dependences between components of the world description. Formally, a domain constraint is a formula

$$Holds(f, s) \leftarrow \pi \tag{8}$$

where the $Holds$ literals in π are only of the form $Holds(f', s)$. The form of the clauses to induce must be changed, because with the normal bias we cannot refer to other effects in the current situation. The use of indirect effects is a key point for learning because some features are better predicted when they are considered *indirect effects* than as direct effects. In this case, forcing all effects as indirect, the new clauses for $light$ and $relay$ are:

```
holds(light,A) :- holds(sw1,A), holds(sw2,A).
-holds(light,A) :- -holds(sw1,A).
-holds(light,A) :- -holds(sw2,A).

holds(relay,A) :- holds(sw1,A), holds(sw3,A).    (a)
-holds(relay,A) :- -holds(sw1,A).
-holds(relay,A) :- -holds(sw3,A).
```

In both cases, we obtained a complete solution. Clause (a) states that the relay is controlled by two switches, i.e., the relay is active

[3] Actions in the form $toggle(sw_i)$ will be denoted as t_i for brevity.

whenever both sw_1 and sw_3 hold simultaneously. Note also that the new clauses *subsume* all effect axioms for $relay$, i.e., they can be used to derive information about its state. Similarly for $light$[4]. Forcing indirect effects in other fluents, e.g., sw_1 and sw_3, may produce strange generalizations or even no compression at all. Fluent sw_2 is more problematic because no complete set of clauses were found, and however some clauses individually were meaningful and subsumed some of the effect axioms.

In general, some fluents can be direct effects of some actions and indirect effects of others. To actually find the right set of axioms, we should induce direct and indirect effects separately, take all induced clauses, and search all subsets of clauses that are complete which a preference criterion (*covering problem*). The worst-case complexity is $O(c^n)$ where c is the number of clauses and n is the largest number of clauses allowed in the solution.

Another possibility we considered is to allow the inductor to determine at each step whether a fluent should be induced as a direct or a indirect effect, thus inducing (possibly) a mix of axioms. Under these conditions the system correctly returned:

```
 holds(sw2,do(t2,A)) :- -holds(sw2,A), -holds(relay,A).
-holds(sw2,A) :- holds(relay,A).
-holds(sw2,do(t2,A)) :- holds(sw2,A).
```

According to the rules, sw_2 is in some cases a direct effect and in other cases an indirect effect. In both cases, a bad selection of the first clauses can cause a "snowballing" effect over subsequent clauses in the cover. In many cases, some helpful information can be extracted from the examples through a procedure to compute so-called influence information [7] among fluents.

Another potentially problematic form of dependency that could arise when learning action theories is *cyclic dependences* between effects. Cyclic definitions can appear even in the description of perfectly normal, well-behaved physical systems. However, care must be taken to ensure that the addition of indirect effects to a SC program does not cause a *mutual recursion*. Let us consider the most classical example:

Example 2 (Gear wheels) *Consider two connected gear wheels together with actions to start (resp. stop) them.* □

Forcing all effects as direct effects, every wheel is considered a direct effect of all actions that affect any of them, i.e., any force causing a wheel to start or stop turning propagates to the rest of wheels (and vic.). When learning indirect effects, our Progol-based algorithm easily induced the two counterparts of a double implication[5].

```
 holds(turn(wheel1),A) :- holds(turn(wheel2),A).
-holds(turn(wheel1),A) :- -holds(turn(wheel2),A).
 holds(turn(wheel2),A) :- holds(turn(wheel1),A).
-holds(turn(wheel2),A) :- -holds(turn(wheel1),A).
```

This theory is a complete solution for learning, i.e., the theory extensionally covers all examples. Unfortunately, this theory is useless as it covers no examples intensionally [10]. In a correct action theory, every fluent must be a direct effect or an indirect effect of any action. In this case, we must introduce some direct effects in the theory. Intuitively we considered $turn(wheel_i)$ as a direct effect of $push(wheel_i)$ and $stop(wheel_i)$.

```
 holds(turn(wheel1),do(push(wheel1),A)).
-holds(turn(wheel1),do(stop(wheel1),A)).
 holds(turn(wheel1),A) :- holds(turn(wheel2),A).
-holds(turn(wheel1),A) :- -holds(turn(wheel2),A).
 holds(turn(wheel2),do(push(wheel2),A)).
-holds(turn(wheel2),do(stop(wheel2),A)).
 holds(turn(wheel2),A) :- holds(turn(wheel1),A).
-holds(turn(wheel2),A) :- -holds(turn(wheel1),A).
```

Now, $turn(wheel_1)$ is a direct effect of $push(wheel_1)$ and an indirect effect of $push(wheel_2)$ –and similarly for $turn(wheel_2)$. However, there is no general rule as to what fluents should be considered direct effects of what actions. In this particular domain, we could just as well to consider $turn(wheel_1)$ totally as a direct effect and $turn(wheel_2)$ as as an indirect effect (or vic.), which makes the induced theory to suffer from the ramification problem, for those fluents where the cycle is broken.

5 Qualification problem

In other than artificial environments, complete knowledge of all relevant facts cannot be assumed. With regard to example 1, when we toggle a switch then, contrary to our expectations, the light may actually not turn on –due to, for instance, a broken bulb, a malfunction of the battery, or loose wiring etc. The presence of just a single "abnormal" example avoids to induce the *general rule* that apply to normal cases. In that case, induced rules should contain the general conditions and all the possible exceptions.

However, the successful execution of actions depends on many more conditions than we are usually aware of. In an extreme case, data includes observations where the prediction of the same action under the same (known) conditions may succeed at one time but fail at another, adding inconsistency to data. Even if BK includes all possible determining factors, the quality of the training set can make an induction algorithm to find a different clause rather than the general one with all the possible exceptions.

In these cases, it is useful to relax the consistency requirement and learn more general clauses that cover a small amount of counterexamples. In dynamic domains, this is particularly interesting as it allows the possibility of discovering *default rules* that describe the most common situations. In so doing, the definitions learned for the positive and negative concepts may overlap, or overlap with the third case (inertia) (Fig. 3). The rectangle represents all situations where an action a is executed[6].

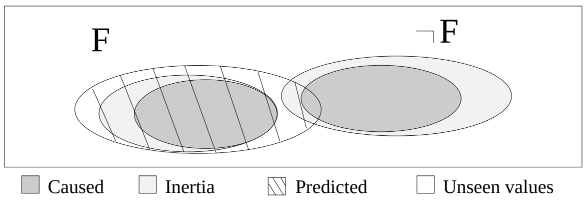

Figure 3. Overlapping concepts

In order to account for exceptions, we introduce for each fluent a unique 'abnormality' predicate ab_i. The ability to assume away, by default, exceptional disqualifications requires some non-monotonic features [4]. In this case, each effect axiom is enhanced by a normality condition, which restricts the axiom to all but abnormal circumstances (using NAF)[7].

[4] A dialect of SC [5] is used with a predicate $Caused$ instead of $Affects$. that allows to express fluent-triggered causal statements (apart from action-triggered ones) which are needed for representing the indirect effects of actions.

[5] Note that, under the normal LP semantics, we need to impose a syntactical restriction (no cycles) that guarantees termination.

[6] In the normal ILP semantics, it is required that the induced program is consistent only with respect to the examples but not necessarily for unseen atoms. Following Lamma et al. the conflict can be resolved by classifying them as undefined through a negative cycle

[7] Truth value must be reified to distinguish when a fluent is abnormal in positive or in negative

$$Holds(f, do(a, s)) \leftarrow Holds(f', s), \ldots, \text{not } Ab(f, \text{true}, a, s)$$

To allow for the induction of default rules, we set an upper bound on the number of negative examples that can be covered by any acceptable clause, such that, induction returns a definition for the concept, consisting of default rules, together with definitions for the abnormality literals. Let us consider the following circuit.

Example 3 (Circuit 2**)** *The circuit consists of the following 15 entities (6 switches, 4 bulbs, 3 resistors and 2 relays): There is only a type of action in this domain, changing the position of switches.*

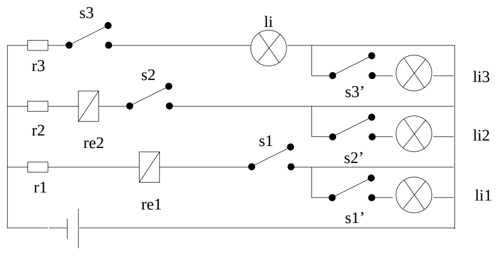

Figure 4. Another electric circuit

□

According to the circuit, the relays, in case of activation, attract the switches located above, thus changing the state of lamps. We included some examples where resistor r_1 failed randomly, hence, when s_1 is closed and r_1 is working abnormally, relay re_1 no longer works correctly. These abnormal examples resulted in the impossibility to learn a definition for li_2. We repeated the learning process relaxing the consistency criterion and found

```
holds(li2,do(t1,A)) :-
          -holds(s1,A), holds(s2p,A), not ab(li2,+,t1,A).
ab(li2,+,t1,do(...,s0)...).
 ...
```

which is the rule previously obtained without abnormal examples. The extension of `abnormal/4` is generated from negative examples covered by the rules and output as a set of ground atoms.

When abnormal examples do not cause inconsistency, we can try to define the circumstances under which a particular abnormality occurs. If exceptions have some common properties, the simple enumeration is not informative and rules about exceptions are useful. This is accomplished by repeating the learning process for ab literals. Positive (resp. negative) examples for `abnormal/4` are obtained from the set of negative (resp. positive) examples covered by the induced rules.

In turn, exceptions to the definitions of `ab(li2)` might be found and so on, thus leading to a hierarchy of exceptions [4] (denoted as ab_i). This procedure would continue specializing until a correct theory is obtained.

$$Ab(f, \text{true}, a, s) \leftarrow Holds(f', s), \ldots, \text{not } Ab1(f, \text{true}, a, s)$$

A question arises as to when default rules should be generated. The strategy above, will necessarily result in fitting the noise in noisy domains. A naive option we have considered is to repeat learning, increasing the number of exceptions allowed, while accuracy improves. A simple heuristics is that the set of exceptions must be smaller than non-exceptions, however more advanced criteria need to be developed that, e.g., allow us to distinguish noisy data from abnormal “examples”.

6 Conclusions

This paper investigates how logic-based induction methods can be adapted for agents to identify the input/output behavior of a dynamical system corresponding to an environment, where the learning output is represented as an action theory. The most distinguishing aspect of action theories is that the specification of actions and their effects should be as intuitive and natural as possible. To this end, Inductive Logic Programming is reformulated using Logic Programming for dynamic systems, and some examples are shown to show its adequation.

Situation Calculus is used as a basic framework, that contributes to a better understanding without requiring complex formalization. Furthermore, the expressivity of SC has been proved sufficient for modeling a wide range of domains and some extensions exist that deal with concurrent actions, continuous change, and so on.

A current research issue is to use more real domains, with a larger number of fluents and actions and with different noise and uncertainty levels. The most challenging domain is *Cognitive Robotics* that is concerned high level cognitive functions of robots that reason, act and perceive in changing, incompletely known, unpredictable environments. In this case, an almost complete description of a robot's actions and its environment is required, which it is a time consuming task [3]. Learning can help construct that description, which will require more recent advances in Action formalisms based on SC to be incorporated.

REFERENCES

[1] C. Baral and S.C. Tran, ‘Relating theories of action and reactive control’, *Linkoping Electronic Articles in Computer Science*, **3**(9), (1998).

[2] M. Gelfond and V. Lifschitz, ‘Representing action and change by logic programs’, *Journal of Logic Programming*, **17**, 301–321, (1993).

[3] V. Klingspor, K. Morik, and A. Rieger, ‘Learning concepts from sensor data of a mobile robot’, *Machine Learning*, **23**, 305–332, (1996).

[4] E. Lamma, F. Riguzzi, and L. Moniz Pereira, ‘Strategies in combined learning via logic programs’, *Machine Learning*, (38), 63–87, (2000).

[5] Fangzhen Lin, ‘Embracing causality in specifying the indirect effects of actions’, in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, ed., C. S. Mellish, Montreal, Canada, (August 1995). Morgan Kaufmann.

[6] D. Lorenzo and R. P. Otero, ‘Learning action theories with causality’, in *Proceedings of the Ninth Inductive Logic Programming Workshop (ILP99), Late Breaking Papers Volume*, eds., P. Flach and S. Džeroski, (1999).

[7] D. Lorenzo and R.P. Otero, ‘Learning action theories as logic programs’, in *International Joint Conference on Declarative Programming*, pp. 383–396, (1999).

[8] J. McCarthy and P.J. Hayes, ‘Some philosophical problems from the standpoint of artificial intelligence’, *Machine Intelligence*, **4**, 463–502, (1969).

[9] S. Muggleton, ‘Inverse entailment and progol’, *New Generation Computing Journal*, (13), 245–286, (1995).

[10] L. Raedt and N. Lavrac, ‘Multiple predicate learning in two ILP settings’, *J. of the IGPL*, **4**(2), 227–254, (1996).

[11] M. Shanahan, *Solving the Frame Problem. A Mathematical Investigation of the Common Sense Law of Inertia.*, The MIT Press, 1997.

[12] M. Shanahan, ‘A logical account of the common sense informatic situation for a mobile robot’, *Linkoping Electronic Articles in Computer Science*, (1999). to appear.

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