Learning STRIPS action descriptions with ILP

David Lorenzo and Ramón P. Otero

Computer Science Dept., Univ. Corunna (Spain)
E-15071 Corunna, Spain
{lorenzo,otero}@dc.fi.udc.es

Abstract. A classical problem in the literature of Machine Learning. is to learn the descriptions of actions and their effects in the form of a set of STRIPS-like operators [7], that are needed for the development of planning techniques. In this work, we introduce a framework to learn STRIPS-like action descriptions based on a logic programming formalization of action domains. Logic Programming provides with a natural and efficient representation where Inductive Logic Programming methods can be applied effectively. Furthermore, learning benefits from the high expressivity of the resulting framework, where STRIPS operators constitute just a subset.

1 Introduction

An important requirement for many intelligent systems is the ability to reason about actions and their effects on the world. Machine Learning methods have been used to learn planning operators [15, 5, 21], in the form of a set of STRIPS-like operators [7]. Some extensions have been considered since then, that are able to cope with more complex effects [4], 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 in the use of implementations of Action Theories based on Logic Programming, i.e., those theories of actions that either use logic programming directly to formalize some aspects of reasoning about actions or those that provide translations to logic programs [8, 3]. Logic Programming provides with a natural and efficient representation where Inductive Logic Programming methods [17] can be applied effectively [12]. An interesting property of these theories is that we can identify subclasses where the logic programming translation has the property of acyclicity and entailment is decidable in general [2]. Furthermore, learning benefits from the high expressivity of the resulting framework, where STRIPS operators constitute just a subset. Unlike classical inductive learning where only propositional learning systems are used, ILP uses Logic Programming as the representational mechanism for hypotheses and observations, and the possibility of including arbitrary logic programs as background knowledge. 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 and avoid the explicit representation of unchanged properties from one particular situation to another (*frame problem*).

This paper is organized as follows. Section 2 briefly describes logic-based formalisms for reasoning about actions and how these can be used for learning STRIPS descriptions by adapting current ILP methods for static domains. In section 3, we describe a prototype based on conventional ILP methods and some experimental results. In section 4 we consider some related work and finally, in section 5, we present some conclusions and outlook future work.

2 Learning to reason about actions

Formalisms for reasoning about actions and change are intended to precisely capture the effects that an *action* produces given the current description of the world and the *preconditions* that specify when the action can be executed. STRIPS-like formalisms are an early attempt to formalize descriptions of actions where the knowledge simply codes the possible answer pairs (action,effect) without more "deductive" features. For instance, the blocks world has the following STRIPS-like description with types and conditional effects.

```
(:action move
         :parameters (?X ?Y ?Z - block)
         :precondition (and (on ?X ?Z) (clear ?X) (clear ?Y)
                            (not (= ?Y ?Z)) (not (= ?X ?Z))
                            (not (= ?X ?Y)) (not (= ?X Table)))
         :effect
         (and (on ?X ?Y) (not (on ?X ?Z))
               (when (not (= ?Z Table)) (clear ?Z))
               (when (not (= ?Y Table)) (not (clear ?Y))))))
```

The above description states the executability conditions of the action $move/3$ (:precondition) the effects (:effect), and the conditional effects, i.e., those effects that depend on the state where the action was executed. There are variants of STRIPS (ADL,etc..) that allow more expressive theories.

The learning task we consider is: given the description of an initial state of the world and a set of observations of a dynamic system, find a STRIPS description (with conditional effects) for each action of a domain that correctly predicts the state of the world after the execution of any sequence of actions starting from any initial state. In action domains, observations for learning are naturally described as *narratives* of actions, consisting of the values of fluents at the initial situation, the actions executed and the effects produced by the actions (Fig. 1).

We will use Logic Programming to represent both the observations and the theory to be learned, which provides with a natural representation where Inductive Logic Programming methods [17] can be applied effectively. Logic Programs

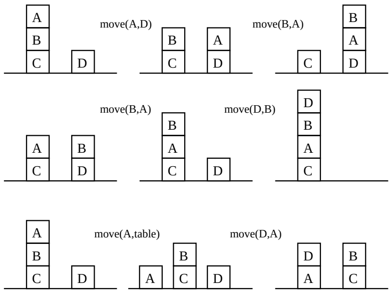

Fig. 1. Narratives in the Blocks world

can be used to represent the effects of actions by importing the ontology of the Situation Calculus [13] or the Event Calculus [9]. We have used a narrative implementation 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 time ($s \geq 0$). This Logic Programming implementation also uses an special predicate *caused/3* [10] and negation as failure to implement inertia. Formally we have:

Definition 1. *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*

$$Caused(f, v, t) \leftarrow Happens(a, t), \pi \tag{1}$$

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).*

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

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

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

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

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

Predicate *caused/3* is used to distinguish when the value of a fluent is caused or holds by inertia. 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.

With respect to the observations, in previous approaches (section 4) each example in, e.g., the blocks world, describes the properties of each block before and after the operation, together with the subject of the operation (i.e., the block to be moved) and its destination, so that each example consists of a separate dataset. This requires an extensive transformation so that the information provided by the narrative, i.e., the sequence of actions, is lost. Unlike this, logic-based languages make the notion of *situation* more central which allow to represent the sequence represented by a narrative, so that learning can be directly applied on the narratives, hence there is no need for preprocessing methods, For instance, the first narrative of Fig. 1 is represented as follows:

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

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.

Positive examples for learning correspond to the ground facts for the predicate *caused/3*. 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 [20].

happens($move(a, d), 1$)	happens($move(b, a), 2$)
holds($on(b, c), 1$)	holds($clear(b), 2$)
...	...

This can be avoided if we include the inertia axiom as part of 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.

With respect to the negative examples for the predicate *caused/3*, it is assumed that all the possible effects in the input narratives are provided, deriving negative information for the predicate *caused/3* by the Closed World Assumption. For this reason, we build negative examples for the predicate *caused/3* for those fluents not affected by actions at each situation.

happens($move(a, d), 1$)	happens($move(b, a), 2$)
caused($on(b, c), f, 1$)	caused($clear(b), f, 2$)
...	...

By doing so, applying the learned rules on the input narratives will not result in postulating new effects not considered initially. This 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 [11].

3 Learning STRIPS with conventional ILP methods

Once narratives are encoded as shown above, conventional ILP methods can be applied to learn a definition for the positive and negative examples of the predicate *caused/3*, that represent the conditional effects of the actions. Additional examples are generated for the predicate *applicable/2* (for those situations where the action executed produced any effect) that correspond to the executability condition of actions. Apart from positive and negative examples, ILP uses so-called *Background Knowledge* (B) consisting basically of predicate definitions so that the definition of a target predicate is built 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 (4,5), and domain-dependent static predicates common to all narratives (for instance, in the blocks world, we have predicates $block/1$, $table/1$ and $diff/2$).

We have developed a prototype $LRAC$ based on the notion of Inverting Entailment (IE) that is the technique used in the system Progol [16]. 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 $applicable(move(c, d), 3)$ is:

```
applicable(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).
```

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 (Fig. 2) 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 once for each action. For conditional effects, i.e., where the conditions for the executability of the action are not enough, we learn the predicate *caused/3* using the predicates *applicable/2*,

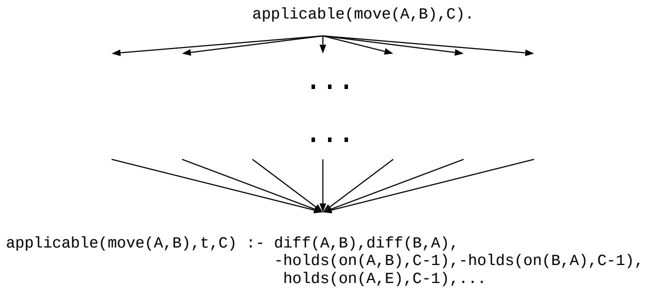

Fig. 2. Top down search

holds/2 and *happens/2* as background knowledge, so that the rule generation phase will be called twice for each fluent, once for the positive concept and once for the negative concept. Rules in the form 1 introduce a bias for the clauses to be learned, where *holds/2* literals in the body can only refer to the previous situation. For instance, the most specific generalization of $caused(on(c,d),t,4)$ is:

```
caused(on(A,B),t,C) :- happens(move(A,B),C), applicable(move(A,B),C),
        -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,B), diff(B,A), 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).
```

For the experimental part we have considered several challenging problems in the literature of reasoning about actions, as well as other domains used in AIPS planning competitions [14]. We used a simple *wander* program that collects data about its actions and the fluents that changed while exploring the environment. The maximum number of hypotheses (nodes) explored during the search was restricted to 5000. This allowed us to keep the execution time under 1 hour. The maximum number of literals in the body of a rule was set to 6, however this limit was never reached. Table 1 shows a summary of the results in some representative domains.

For instance, in the blocks world, from 48 narratives of length 6, *LRAC* learned the following clauses for the predicate *applicable/2*:

```
applicable(move(A,B),C) :- diff(A,B),-holds(on(A,B),C),
                           holds(clear(A),C),holds(clear(B),C).
applicable(move(A,B),C) :- diff(A,B),-holds(on(A,B),C),
                           holds(clear(A),C),table(B).
```

The above clauses deal separately with the cases where a block is moved onto other block or onto the table. With respect to the predicate *caused/3*, *LRAC* learned:

```
caused(on(A,B),t,C) :- happens(move(A,B),C),applicable(move(A,B),C-1).
```

Domain	examples	rules	time	nodes
blocks world	(8,8)	(4,2)	36.64	918
hanoi towers	(7,6)	(2,2)	135.29	3951
logistics (AIPS)	(22,16)	(5,3)	565.96	69
robot domain (AIPS)	(24,10)	(7,1)	6.72	366
manhattan world (AIPS)	(4,19)	(4,2)	2170.44	2585

Table 1. Summary of training data: domain, number of narratives and length of each narrative, number of axioms learned for the predicate *caused/3* and *applicable/2*, time used for training (in seconds) and number of nodes explored.

```
 caused(on(A,B),f,C) :- happens(move(A,D),C),applicable(move(A,D),C-1),
                        holds(on(A,B),C-1).
caused(clear(A),t,B) :- happens(move(C,D),B),applicable(move(C,D),B-1),
                        holds(on(C,A),B-1).
caused(clear(B),f,A) :- happens(move(C,B),A),applicable(move(C,B),A-1),
                        not table(B).
```

so that both *clear*/1 and the negative value of *on*/2 are conditional effects of the action[1]. The above theory has a straightforward translation into a STRIPS theory with conditional effects.

4 Related work

Many approaches exist to learning domain knowledge for planning [20, 15, 5, 19, 21, 18, 6, 1]. They learn or refine operator effects through experimentation or from traces from experts' problem solving. In many of them, most emphasis is put on the autonomy of learning, where planning, learning and exploratory steps are interleaved [4, 20, 19, 21, 5], whereas others combine a phase of exploration, e.g., with a *wander* program that records observations [1, 18], and a phase of generalization carried out by inductive learning systems. The difference is that the latter approaches consider the learning process and the exploratory process as two separate processes, so that the learning algorithm has absolutely no control over the sample of labeled examples drawn, i.e., they do not perform actions to control the system, and so learning is not really active. Our approach falls into this second approach.

Gil [5] presents an approach to refining a planner's incomplete knowledge base consisting of STRIPS-style operators, that uses exploration and experimentation, so that an inductive learner is trained on the instances of successful and unsuccessful operators and the states where the operators were chosen to fill in missing preconditions and effects. Wang's OBSERVER system [21] is able to work from an empty knowledge base, using traces from experts' problem solving to fill in the empty operator descriptions. TRAIL [4] extends this work by allowing disjunctions in the preconditions, noise in an action's effects, and

[1] The negative value of *on*/2 is usually not considered a conditional effect of the action *move*/3 that also includes the previous location of the block.

also durative actions. TRAIL uses so-called teleo-operators (TOP) [4] that are an action representation different from STRIPS operators, which are appropriate to reactive agents, but makes the results of learning very dependent on the particular formalisms and difficult to transfer.

5 Conclusions

We have developed a framework to learn STRIPS-like action descriptions based on a logic programming formalization of action domains where ILP methods can be applied effectively. The basic model can be extended to deal with indirect effects of actions and with concurrent actions [11], or to deal with input narratives that are not completely specified. Further work needs also 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.

References

1. N. Balac, D.M. Gaines, and D. Fisher. Using regression trees to learn action models. In *IEEE Systems, Man and Cybernetics Conference*, 2000.
2. C. Baral. Relating logic programming theories of actions and partial order planning. *Annals of Math and AI*, 21(2–4), 1997.
3. C. Baral and M. Gelfond. Reasoning about effects of concurrent actions. *Journal of Logic Programming*, 31(1–3):85–117, 1997.
4. S. Benson. Inductive learning of reactive action models. In *Proc. 12th International Conference on Machine Learning*, pages 47–54. Morgan Kaufmann, 1995.
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. M. desJardins. Knowledge development methods for planning systems. In *Planning and Learning: On to Real Applications: Papers from the 1994 AAAI Fall Symposium*, pages 34–40. AAAI Press, Menlo Park, California, 1994.
7. 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.
8. M. Gelfond and V. Lifschitz. Representing Actions and Change by Logic Programs. *Journal of Logic Programming*, 17:301–322, 1993.
9. R. Kowalski and M. Sergot. A logic-based calculus of events. *New Generation Computing*, 4:67–95, 1986.
10. 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.
11. 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.

12. 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.
13. J. McCarthy and P.J. Hayes. Some philosophical problems from the standpoint of artificial intelligence. *Machine Intelligence*, 4:463–502, 1969.
14. D. McDermott. The 1998 ai planning systems competition. *AI Magazine*, 21(2):35–55, 2000.
15. 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.
16. S. Muggleton. Inverse entailment and Progol. *New Generation Computing, Special issue on Inductive Logic Programming*, 13(3-4):245–286, 1995.
17. S. Muggleton and L. De Raedt. Inductive logic programming: Theory and methods. *Journal of Logic Programming*, 19/20:629–679, 1994.
18. T. Oates and P. R. Cohen. Learning planning operators with conditional and probabilistic effects. In *Planning with Incomplete Information for Robot Problems: Papers from the 1996 AAAI Spring Symposium*, pages 86–94. AAAI Press, Menlo Park, California, 1996.
19. D. J. Pearson. Learning procedural planning knowledge in complex environments. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence and the Eighth Innovative Applications of Artificial Intelligence Conference*, pages 1401–1401, Menlo Park, 1996. AAAI Press / MIT Press.
20. W. M. Shen. *Autonomous Learning from the Environment.* Computer Science Press, 1994.
21. 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.