Inductive Corrections of Action Descriptions

Marcello Balduccini

Computer Science Department
Texas Tech University
Lubbock, TX 79409 USA
marcello.balduccini@ttu.edu

Abstract. In this paper, we show how A-Prolog can be used to perform *incremental, inductive learning* of the behavior of dynamic domains from (possibly incomplete) observations. The learning module deals with both *direct and indirect effects of actions*, and is not affected by the frame problem. Moreover, the module can be combined with planning and diagnostic modules, allowing for its integration in autonomous agents.

1 Introduction

A-Prolog, the language of logic programs under the answer set semantics [2], has been recently used to design sophisticated planning and diagnostic modules for a fairly large class of domains. To the best of our knowledge, however, no attempt was made to *design an A-Prolog software component capable of inductive learning of the behavior of dynamic domains.* Our goal in this paper is to describe such a software component. At this stage of the development, our main concerns are assessing the feasibility of the approach, its simplicity and flexibility, as well as the ability to integrate the module with other reasoning components.

2 The Learning Task

The learning component takes as input an action description and a history and computes possible corrections of the action description that agree with the history. Because of space restrictions, in this paper we write the laws directly in the logic programming encoding used by our learning module. Hence, a *dynamic law* is encoded by a set of atoms $d\_law(\varpi)$, $head(\varpi, l_0)$, $action(\varpi, a_e)$, $prec(\varpi, l_1)$,$prec(\varpi, l_2)$, ..., $prec(\varpi, l_n)$, where ϖ is the name of the law, l_i's are terms denoting fluent literals (names of relevant properties and their negations), and a_e is an action. The intuitive reading is "if a_e is executed in a state in which $l_1, \ldots, l_n$ hold, l_0 holds at the next step." The terms are possibly *non-ground*, with the restriction that every variable that occurs in l_i or a_e must also occur in ϖ. A *state constraint* is represented by a set of atoms $s\_law(\varpi)$, $head(\varpi, l_0)$, $prec(\varpi, l_1)$, ..., where ϖ and l_i's are as above. The intuitive reading is "if $l_1, \ldots, l_n$ hold, then l_0 must hold as well (at the same step)." *Observations on fluents* are statements $obs(l, s)$ stating that fluent literal l was observed to hold at step s; *observations on actions* are statements $hpd(a_e, s)$, stating that action a_e was observed to occur at step s. An *action description* AD is a set of dynamic laws and state constraints[1]. A *recorded history* up to step cT, H^{cT}, is a set of observations on fluents and on actions up to step cT. A *domain description* is a pair $\mathcal{D} = \langle AD, H^{cT} \rangle$. The semantics of $\mathcal{D}$ is given by the set of A-Prolog rules, Π:

% If the preconditions of a state constraint hold, its head holds.
1. $h(L,T) \quad \leftarrow s\_law(W), head(W,L), prec\_h(W,T).$
% If the preconditions hold and the action occurred, the head holds at the next step.
2. $h(L,T+1) \leftarrow d\_law(W), head(W,L), prec\_h(W,T), action(W,A), o(A,T).$
% The inertia axiom: normally, things stay as they are.
3. $h(L,T+1) \leftarrow h(L,T1), \text{not } h(\overline{L},T2).$
% Observations must match the predictions.
4. $\qquad \leftarrow obs(L,T), \text{ not } h(L,T).$
% Observations on the initial state and actions are taken as-is.
5. $o(A,T) \quad \leftarrow hpd(A,T).$
6. $h(L,0) \quad \leftarrow obs(L,0).$

where $h(L,T)$ (resp., $o(A,T)$) says that L holds at T (resp., A occurs at T). Relation $prec\_h(W,T)$ holds when all the preconditions of W hold at T. Rule 4 causes inconsistency if unexpected observations are detected. We say that H^{cT} is a *symptom* (for $\mathcal{D}$) if $AD \cup H^{cT} \cup \Pi$ is inconsistent. In the context of this work, a symptom indicates that the action description needs to be modified. A *modification* for AD is a set of atoms encoding new causal laws, and of atoms encoding additional preconditions for the laws already in AD.

Definition 1. *An* inductive correction *of AD for symptom H^{cT} is a modification $\mathcal{M}$ for AD such that $H^{cT} \cup AD \cup \mathcal{M} \cup \Pi$ is consistent.*

From the point of view of learning, rule (4) above guarantees that consistency is achieved only when all the observations (both positive and negative) are explained, i.e. for every $obs(l,s) \in H^{cT}$, $H^{cT} \cup AD \cup \mathcal{M} \cup \Pi \models h(l,s)$. The A-Prolog learning module is based on a generate-and-test approach in which modifications are generated and tested to see if they are inductive corrections. In A-Prolog modifications can be easily generated by using *choice rules*. For example, given $a(1)$, $a(2)$, the rule $1\{p(Y) : a(Y)\}1$ intuitively generates two alternative selections, $\{p(1)\}$ and $\{p(2)\}$, of cardinality 1. Each selection intuitively leads to a separate *answer set*, containing the selection and its consequences.[2]

The main issue in writing the A-Prolog learning module is that non-ground terms have to be given a ground name. For simplicity, we denote a term $t(V_1, \ldots, V_n)$ by $t_{[\nu_1,\ldots,\nu_n]}$.[3] The latter is called the *groundification* of the former. The link between the two is encoded by relation $gr(t(V_1,\ldots,V_n), t_{[\nu_1,\ldots,\nu_n]})$. Relation $litname(L)$ (resp., $actname(A)$) holds for ground and groundified fluent literals (resp., actions). Relation

[1] In the full paper, we also allow impossibility conditions.
[2] For a mathematical definition of the semantics, please refer to [2].
[3] Better naming schemes can be designed, but are out of the scope of this paper.

$new\_lawname(W)$ holds if W is the groundification of a name of a law that does not occur in the laws of AD. We also need a way to simulate the grounding of laws. This is useful when we are given an atom $prec(\varpi_{[\nu_1,\ldots,\nu_n]}, l_{[\nu_1,\ldots,\nu_m]})$ and want to instantiate $l_{[\nu_1,\ldots,\nu_n]}$ for the ground instance $\varpi(c_1,\ldots,c_n)$ of the law. To this purpose, we use relation $denotes(l_{[\nu_1,\ldots,\nu_m]}, \varpi_{[\nu_1,\ldots,\nu_n]}, \varpi(c_1,\ldots,c_n), l(c_1,\ldots,c_m))$. Notice that all of the above relations can be automatically pre-computed.

The learning module, $\mathcal{L}$, consists of the following rules[4]:

% Generate new dynamic laws or state constraints as needed.
1. $0\{missing\_law(W) : new\_lawname(W)\}\infty.$
2. $1\{d\_law(W'), s\_law(W')\}1 \leftarrow missing\_law(W), gr(W,W').$

% Select groundified head, action, and preconditions as needed.
3. $1\{needs(W, head(L)) : litname(L)\}1 \leftarrow missing\_law(W).$
4. $1\{needs(W, action(A)) : actname(A)\}1 \leftarrow missing\_law(W), gr(W,W'), d\_law(W').$
5. $0\{needs(W, prec(N,L)) : litname(L)\}1 \leftarrow gr(W,W'), prec\_index(N).$

% Map groundified terms into their groundings.
6. $head(W,L) \leftarrow needs(W', head(L')), denotes(L',W',W,L).$
7. $action(W,A) \leftarrow needs(W', action(A')), denotes(A',W',W,A).$
8. $prec(W,N,L) \leftarrow needs(W', prec(N,L')), denotes(L',W',W,L).$

We say that the atoms $needs(W,X)$ above *define the modification* that can be extracted from their second arguments.

Theorem 1. *Let $\mathcal{D}$ be domain description, $\mathcal{S}$ be a symptom, and $\mathcal{M}$ a modification. An answer set, A, of $H^{cT} \cup AD \cup \Pi \cup \mathcal{L}$ defines $\mathcal{M}$, if and only if $\mathcal{M}$ is an inductive correction of AD for $\mathcal{S}$.*

Notice that our approach deals with both direct and indirect effects of actions. Also, because the inertia axiom is built in Π, the learning module can generate action descriptions that do not have the frame problem [3]. As preconditions can be added to existing laws, the process can be viewed as *incremental*. Extensions of A-Prolog also allow to specify heuristics and preferences to improve the quality of the learning process. Finally, a comparison with [1] and derived work shows that $\mathcal{L}$ shares the *same encoding* of the existing planning and diagnostic modules. Thus, it can be integrated in agent architectures based on the cited approach.

References

1. Chitta Baral and Michael Gelfond. Reasoning Agents In Dynamic Domains. In *Workshop on Logic-Based Artificial Intelligence*, pages 257–279. Kluwer Academic Publishers, Jun 2000.
2. Michael Gelfond and Vladimir Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, pages 365–385, 1991.
3. Ramon Otero. Induction of the effects of actions by monotonic methods. In *Proceedings of the 13th International Conference on Inductive Logic Programming, ILP 03*, number 2835 in Lecture Notes in Artificial Intelligence (LNCS), pages 299–310, 2003.

[4] A few rules were omitted to save space. The full paper contains the complete module.