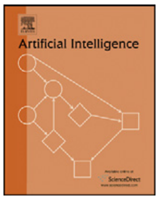
Artificial Intelligence 174 (2010) 1540–1569

Contents lists available at ScienceDirect

Artificial Intelligence

www.elsevier.com/locate/artint



Learning complex action models with quantifiers and logical implications

Hankz Hankui Zhuo[a,b], Qiang Yang[b,*], Derek Hao Hu[b], Lei Li[a]

[a] *Department of Computer Science, Sun Yat-sen University, Guangzhou, China, 510275*
[b] *Department of Computer Science and Engineering, Hong Kong University of Science and Technology, Clearwater Bay, Kowloon, Hong Kong*

ARTICLE INFO

Article history:
Received 2 January 2010
Received in revised form 4 September 2010
Accepted 5 September 2010
Available online 29 September 2010

Keywords:
Action model learning
Machine learning
Knowledge engineering
Automated planning

ABSTRACT

Automated planning requires action models described using languages such as the **P**lanning **D**omain **D**efinition **L**anguage (PDDL) as input, but building action models from scratch is a very difficult and time-consuming task, even for experts. This is because it is difficult to formally describe all conditions and changes, reflected in the preconditions and effects of action models. In the past, there have been algorithms that can automatically learn simple action models from plan traces. However, there are many cases in the real world where we need more complicated expressions based on universal and existential quantifiers, as well as logical implications in action models to precisely describe the underlying mechanisms of the actions. Such complex action models cannot be learned using many previous algorithms. In this article, we present a novel algorithm called LAMP (**L**earning **A**ction **M**odels from **P**lan traces), to learn action models with quantifiers and logical implications from a set of observed plan traces with only partially observed intermediate state information. The LAMP algorithm generates candidate formulas that are passed to a Markov Logic Network (MLN) for selecting the most likely subsets of candidate formulas. The selected subset of formulas is then transformed into learned action models, which can then be tweaked by domain experts to arrive at the final models. We evaluate our approach in four planning domains to demonstrate that LAMP is effective in learning complex action models. We also analyze the human effort saved by using LAMP in helping to create action models through a user study. Finally, we apply LAMP to a real-world application domain for *software requirement engineering* to help the engineers acquire software requirements and show that LAMP can indeed help experts a great deal in real-world knowledge-engineering applications.

1. Introduction

Automated planning systems achieve goals by producing sequences of actions from the given action models that are provided as input [14]. A typical way to describe the action models is to use action languages such as the **P**lanning **D**omain **D**efinition **L**anguage (PDDL) [13,11,14] in which one can specify the precedence and consequence of actions. A traditional way of building action models is to ask domain experts to analyze a task domain and manually construct a domain description that includes a set of complete action models. Planning systems can then proceed to generate action sequences to achieve goals.

However, it is very difficult and time-consuming to manually build action models in a given task domain, even for experts. This is a typical problem of the knowledge-engineering bottleneck, where experts often find it difficult to articulate their experiences formally and completely. Because of this, researchers have started to explore ways to reduce the human

* Corresponding author.
E-mail addresses: zhuohank@mail.sysu.edu.cn (H.H. Zhuo), qyang@cse.ust.hk (Q. Yang), derekhh@cse.ust.hk (D.H. Hu), lnslilei@mail.sysu.edu.cn (L. Li).

0004-3702/$ – see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.artint.2010.09.007

effort of building action models by learning from observed examples or plan traces. Some researchers have developed methods to learn action models from complete state information before and after an action in some example plan traces [4, 15,31,50]. Others, such as Yang et al. [51,39] have proposed to learn action models from plan examples with only incomplete state information. Yang et al. [51,52] have developed an approach known as *Action Relation Modeling System* (ARMS) to learn action models in a STRIPS (**ST**anford **R**esearch **I**nstitute **P**roblem **S**olver) [10] representation using a weighted MAXSAT-based (Maximum Satisfiability) approach. Shahaf et al. [39] have proposed an algorithm called *Simultaneous Learning and Filtering* (SLAF) to learn more expressive action models using consistency-based algorithms.

Despite the success of these learning systems, in the real world, there are many applications where actions should be expressed using a more expressive representation, namely, quantifiers and logical implications. For instance, consider the case where there are different *cases* in a *briefcase*[1] planning domain, such that a briefcase should not be moved to a place where there is another briefcase with the same color. We can model the action *move* in PDDL as follows.[2]

```
action:   move(?c1 - case ?l1 ?l2 - location)
pre:      (:and (at ?c1 ?l1)
            (forall ?c2 - case (imply (samecolor ?c2 ?c1)(not (at ?c2 ?l2)))))
effect:   (:and (at ?c1 ?l2) (not (at ?c1 ?l1)))
```

That is, if we want to move the case *c1* from the location *l1* to *l2*, *c1* should be at *l1* first, and every other case *c2* whose color is the same with *c1* should not be at *l2*. After the action *move*, *c1* will be at *l2* instead of at *l1*. Likewise, consider a pilot could not fly to a place where there are enemies. We can model the action model *fly* as follows.

```
action:   fly(?p1 - pilot ?l1 ?l2 - location)
pre:      (:and (at ?p1 ?l1)
            (forall ?p2 - person (imply (enemy ?p2 ?p1)(not (at ?p2 ?l2)))))
effect:   (:and (at ?p1 ?l2) (not (at ?p1 ?l1)))
```

We can see that in these examples, we need universal quantifiers as well as logical implications in the precondition part of the action to precisely represent this action and compress the action model in a compact form.

As another example, consider a driver who intends to drive a train. Before he can start, he should make sure all the passengers have gotten on the train. After that, if there is a seat vacant, then he can start to drive the train. We represent this *drive-train* action model in PDDL as follows.

```
action:   drive-train(?d - driver ?t - train)
pre:      (free ?d) (forall ?p - passenger (in ?p ?t))
effect:   (:and (when (exist ?s - seat (vacant ?s)) (available ?t))
                (driving ?d ?t)(not (free ?d)))
```

That is, if a driver *?d* makes sure all the passengers *?p* are in the train *?t* and is free at that time, then he can drive the train *?t*. Furthermore, if there is a seat *?s* vacant, as a consequence of this action *drive-train*, the train will be set as available to show that more passengers can take this train. Besides, the driver *?d* will be in the state of driving the train, i.e., *(driving ?d ?t)*, and not free. Such an action model needs a universal quantifier in describing its preconditions and an existential quantifier for the condition "*(exist ?s - seat (vacant ?s))*" of the conditional effect "*(when (exist ?s - seat (vacant ?s))(available ?t))*". More examples that require the use of quantifiers and logical implications can be found in many action models in recent International Planning Competitions, such as the domains in IPC-5[3]: *trucks*, *openstacks*, etc. These complex action models can be represented by PDDL, but cannot be learned by existing algorithms proposed for action model learning.

Our objective is to develop a new algorithm for learning complex action models with quantifiers (including conditional effects) and logical implications, from a collection of given example plan traces. The input of our algorithm includes: (1) a set of observed plan traces with partially observed intermediate state information between actions; (2) a list of action headings, each of which is composed of an action name and a list of parameters, but is not provided with preconditions or effects; (3) a list of predicates along with their corresponding parameters. Our algorithm is called LAMP (***L**earn **A**ction **M**odels from **P**lan traces*), which outputs a set of action models with quantifiers and implications. These action models 'summarizes' the plan traces as much as possible, and can be used by domain experts, who need to spend only a small amount of time, in revising parts of the action models that are incorrect or incomplete, before finalizing the action models for planning usage.

Compared to many previous approaches, our main contributions are: (1) LAMP can learn quantifiers that conform to the PDDL definition [13,11], where the latter article shows that action models in PDDL can have quantifiers. (2) LAMP can learn action models with implications as preconditions, which improves the expressiveness of learned action models. We require

[1] http://www.informatik.uni-freiburg.de/~koehler/ipp/pddl-domains.tar.gz.
[2] A symbol with a prefix "?" suggests that the symbol is a variable; e.g. "?c1" suggests that "c1" is a variable that can take on certain constants as values.
[3] http://zeus.ing.unibs.it/ipc-5/domains.html.