Action Models and their Induction

Michal Čertický, *Comenius University, Bratislava*

certicky@fmph.uniba.sk

March 5, 2013

Abstract

By action model, we understand any logic-based representation of effects and executability preconditions of individual actions within a certain domain. In the context of artificial intelligence, such models are necessary for planning and goal-oriented automated behaviour. Currently, action models are commonly hand-written by domain experts in advance. However, since this process is often difficult, time-consuming, and error-prone, it makes sense to let agents learn the effects and conditions of actions from their own observations. Even though the research in the area of action learning, as a certain kind of inductive reasoning, is relatively young, there already exist several distinctive action learning methods. We will try to identify the collection of the most important properties of these methods, or challenges that they are trying to overcome, and briefly outline their impact on practical applications.

1 Introduction

Reasoning about actions is an important aspect of commonsense reasoning, which served as a motivation behind some of the recent nonmonotonic logic formalisms and planning languages (Eiter et al., 2000; Giunchiglia and Lifschitz, 1998; McDermott et al., 1998; Pednault, 1989; Ginsberg and Smith, 1988). Intelligent and flexible goal-oriented automated behaviour and planning tasks require *knowledge about domain dynamics*, describing how certain actions affect the world. Such knowledge is in artificial systems referred to as *action model*.

In general, the action model can be seen as a double $\langle D, P \rangle$, where D is a representation of *domain dynamics* (*effects* and *executability preconditions* of every possible action) in any logic-based language, and P is a probability function defined over the elements of D. This probability expresses either the likelihood of certain action's effect, or our confidence in this piece of knowledge.

Typically, these action models are hand-written by domain experts. In many situations however, we would like to be able to induce such models automatically, since hand-writing them is often a difficult, time-consuming and error-prone task (especially in complex environments). In addition to that, every time we are confronted with new information, we need to do (often problematic) knowledge revisions and modifications.

An agent (artificial or living) capable of learning action models possesses some degree of environmental independence (he can be deployed into various environments, where he would learn local causal dependencies and consequences of his actions).

The inductive process of automatic construction and subsequent improvement of action models, based on sensory observations, is called *action learning*. In recent years, several action learning methods have been introduced. They take various approaches and employ a wide variety of tools from many areas of artificial intelligence and computer science (Amir and Chang, 2008; Yang et al., 2007; Balduccini, 2007; Certicky, 2012; Mourão et al., 2010; Zettlemoyer et al., 2005). In this paper, we will describe a collection of *interesting properties*, or *fundamental challenges* that any action learning method might, or might not be able to overcome.

2 Usability in Partially Observable Domains

Every domain is either fully, or partially observable. As an example of a *fully observable domain* let us consider a game of chess. Both players (agents) have a full visibility of all the features of their domain - in this case the configuration of the pieces on the board. Such configuration is typically called a *world state*. On the other hand, by *partially observable domain* we understand any environment, in which agents have only limited observational capabilities - in other words, they can see only a small part of the state of their environment (world states are partially observable). Real world is an excellent example of a partially observable domain. Agents of the real world (for example humans) can only observe a small part of their surroundings: they can only hear sounds from their closest vicinity (basically several meters, depending on how loud the sounds are), see only objects that are in their direct line of sight (given the light conditions are good enough), etc.

An action learning method is *usable in partially observable domains* only if it is capable of producing useful action models, even if the world states are not fully observable.

Learning the action models in partially observable domains is in principle more difficult task, since we do not observe some of the changes happening in the world after the execution of actions. To induce

a causal link between the action and its effect, we need to observe this effect. However, in partially observable domains, this observation may be available later or not at all, making the learning slower and resulting models less precise.

3 Learning Probabilistic Action Models

There are two ways of modelling a domain dynamics (creating action models), depending on whether we want the randomness to be present or not. An action model is *deterministic*, if actions it describes have all a unique set of always successful effects. In other words, the probabilistic function P assigns the uniform probability of 1 to all the elements of D.

Conversely, in case of a *probabilistic* (or stochastic) action models, effects have a set of possible *outcomes* with non-uniform probabilistic distrubution. Let us clarify this concept using a simple toy domain called Blocks World, discussed extensively (among others) in (Nilsson, 1982; Russell and Norvig, 2003; Gupta and Nau, 1992; Slaney and Thibaux, 2001).

The Blocks World domain consists of a finite number of blocks stacked into towers on a table large enough to hold them all. The positioning of towers on the table is irrelevant. Agents can manipulate this domain by moving blocks from one position to another. Action model of the simplest Blocks World versions is composed of only one action $move(B, P_1, P_2)$. This action merely moves a block B from position P_1 to position P_2 (P_1 and P_2 being either another block, or the table).

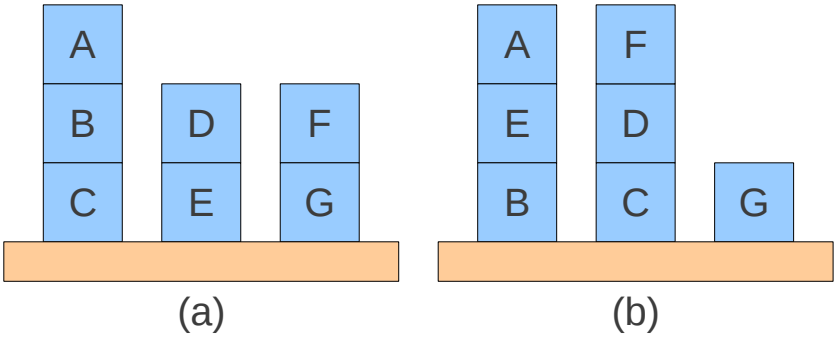

Figure 1: Two different world states in Blocks World domain.

Deterministic representation of such action would look something like this:

$$\begin{aligned}
&\text{Name \& parameters:}\\
&\qquad move(B, P_1, P_2)\\
&\text{Preconditions:}\\
&\qquad \{on(B, P_1), free(P_1), free(P_2)\}\\
&\text{Effects:}\\
&\qquad \{\neg on(B, P_1), on(B, P_2)\}
\end{aligned}$$

Our action is defined by its name, preconditions, and a unique set of effects $\{\neg on(B, P_1), on(B, P_2)\}$, all of which are applied each time the action is executed. This basically means, that every time we perform an action $move(B, P_1, P_2)$, the block B will cease to be at position P_1 and will appear at P_2 instead. In a simple domain like Blocks World, this seems to be sufficient.

In the real world however, the situation is not so simple, and our attempt to move the block can have different outcomes:

$$\begin{aligned}
&\text{Name \& parameters:}\\
&\qquad move(B, P_1, P_2)\\
&\text{Preconditions:}\\
&\qquad \{on(B, P1), free(P_1), free(P_2)\}\\
&\text{Effects:}\\
&\qquad \begin{cases}
0.8: & \neg on(B, P_1), on(B, P_2)\\
0.1: & \neg on(B, P_1), on(B, table)\\
0.1: & nochange
\end{cases}
\end{aligned}$$

This representation of our action defines the following probabilistic distribution over three possible outcomes:

1. 80% chance that block B indeed appears at P_2 instead of P_1,
2. 10% chance that block B falls down on the table,
3. 10% chance that we fail to pick it up and nothing happens.

We can easily see, that probabilistic action models are better suited for describing real-world domains, or complex simulations of non-deterministic nature, where agent's sensors and effectors may be imprecise and actions can sometimes lead to unpredicted outcomes.

The main difficulty in learning probabilistic action models lies in their size. Space complexity of such

models tends to be considerably higher, and learning algorithms need to be able to distinguish relevant outcomes and ignore the others.

4 Dealing with Action Failures and Sensoric Noise

In some cases we prefer learning deterministic action models in stochastic domains. (Recall, that action models are used for planning. Planning with probabilistic models is computationally harder, which makes it unusable in some situations.) Therefore we need an alternative way of dealing with nondeterministic nature of our domain. There are two sources of problems that can arise in this setting:

4.1 Action Failures

As we noted in section 3, actions in non-deterministic domains can have more than one outcome. In a typical situation though, each action has one outcome with significantly higher probability than the others. In case of action $move(B, P_1, P_2)$ from Blocks World, this *expected outcome* was actually moving a block B from position P_1 to P_2. Then if after the execution the block was truly at position P_2, we considered the action successful. If the action had any other outcome, it was considered unsuccessful - we say that the action *failed*.

From the agent's point of view, action failures pose a serious problem, since it is difficult for him to decide whether given action really failed (due to some external influence), or the action was successfull, but his expectations about the effects were wrong (if his expectations were wrong, he needs to modify his action model accordingly).

4.2 Sensoric Noise

Another source of complications is so-called *sensoric noise*. In real-world domains, we are typically dealing with sensors that have limited precision. This means, that the observations we get do not necessarily correspond to the actual state of the world.

Even when agent's action is successful, and the expected changes occur, he may observe the opposite. From the agent's point of view, this problem is similar to the problem with action failures. In this case he needs to solve the dilemma, whether his expectations were incorrect, or the observation was imprecise.

In addition to that, sensoric noise can cause one more complication of a technical nature: If the

precision of the observations is not guaranteed, even a single observation can be internally inconsistent. Action learning methods based on a computational logic sometimes fail to deal with this fact.

5 Learning both Preconditions and Effects

Since the introduction of the first planning language STRIPS (Fikes and Nilsson, 1971) in early 70's, a common assumption is, that actions have some sort of *preconditions* and *effects*.

Preconditions[*] define what must be established in a given world state before an action can even be executed. Looking back at Blocks World, the preconditions of action $move(B, P_1, P_2)$ require both positions P_1 and P_2 to be free (meaning that no other block is currently on top of them). Otherwise, this action is considered inexecutable.

Effects[†] simply specify what is established after a given action is executed, or in other words, how the action modifies the world state.

Some action learning approaches either produce effects and ignore preconditions, or the other way around. They are therefore incapable of producing complete action model from the scratch, and thus are usable only in situations when some partial hand-written action model is provided. In general, it is good to avoid the necessity to have any prior action model.

6 Learning Conditional Effects

Research in the field of planning languages has shown, that expressive power of early (STRIPS-like) representations is susceptible to be improved by addition of so-called conditional effects. This results from the fact, that actions, as we usually talk about them in natural language, have different effects in different world states.

Consider a simple action of person P drinking a glass of beverage B - $drink(P, B)$. Effects of such action would be (in natural language) expressed by following sentences:

- P will cease to be thirsty.
- If B was poisonous, P will be sick.

[*]Preconditions are sometimes called *executability conditions* or *applicability conditions* - especially when we formalise actions as operators over the set of world states.

[†]Effects are sometimes called *postconditions* - primarily in the early publications in STRIPS-related context.

We can see, that second effect (P becoming sick) only applies *under certain conditions* (only if B was poisonous). We call effects like these *conditional effects*.

Early planning languages did not support conditional effects. Of course, there was a way to express aforementioned example, but we needed split it into two separate actions with different sets of preconditions:
$drink\_if\_poisonous(P, B)$ and $drink\_if\_not\_poisonous(P, B)$.

Having a support for conditional effects thus allows us to express domain dynamics by lower number of actions, making our representation less space consuming and more elegant. Several state-of-the art planning languages provide the apparatus for defining conditional effects - see the following example:

STRIPS extensions like Action Description Language (ADL) (Pednault, 1989) or Planning Domain Definition Language (PDDL) (McDermott et al., 1998) express the effects of $drink(P, B)$ action in the following manner:

```
:effect     (not (thirsty ?p))
:effect     (when (poisonous ?b) (sick ?p))
```

Definition of same two effects in fluent-based languages like $\mathcal{K}$ (Eiter et al., 2000) on the other hand, employs the notion of so-called *dynamic laws*:

```
caused -thirsty(P) after drink(P,B).
caused sick(P) after poisonous(B), drink(P,B).
```

Aside from creating more elegant and brief action models, the ability to learn conditional effects provides one important advantage: It allows for more convenient input form from our sensors. If we were unable to work with conditional effects, our sensors would have to be able to observe and interpret a large number of actions like $drink\_if\_poisonous(P, B)$ or $drink\_if\_not\_poisonous(P, B)$. However, if our action model supports conditional effects, the sensors only need to work with a smaller number of more general actions like $drink(P, B)$.

7 Online Algorithms and Tractability

As mentioned in the introduction, the action learning methods employ various tools from several areas of computer science and artificial intelligence. Since our focus lies on the artificial agents, and their ability to learn action models, either these tools themselves, or their actual objectification is algorithmic in nature. It is therefore needed to take the computational complexity and the actual running speed of

used algorithms into account. We say, that algorithms that run fast enough for their output to be useful are called *tractable* (Hopcroft, 2007).

Additionally, the algorithms whose input is served one piece at a time, and upon receiving it, they have to take an irreversible action without the knowledge of future inputs, are called *online* (Borodin and El-Yaniv, 1998).

For the purposes of action learning we prefer using online algorithms, which run once after every observation. Agent's newest observation is served as the input for the algorithm, while there is no way of knowing anything about future or past observations. Algorithm simply uses this observation to modify agent's knowledge (action model). Since the input of such algorithm is relatively small, tractability is usually not an issue here.

If we, on the other hand, decided to use offline algorithms for action learning, we would have to provide the whole history of observations on the input. Algorithms operating over such large data sets are prone to be intractable.

Since online algorithms are designed to run repeatedly during the "life" of an agent, he has some (increasingly accurate) knowledge at his disposal at all times. Offline action learning algorithms are, on the other hand, designed to run only once, after the agent's life, which makes them unusable in many applications.

There is however a downside to using online algorithms for action learning. Recall, that with online algorithms, the complete history of observations is not at our disposal, and we make an irreversible change to our action model after each observation. This change can cause our model to become inconsistent with some of the previous (or future) observations. This also means, that the precision of induced action models depends on the ordering of the observations. Online algorithms are therefore potentially less precise than their offline counterparts. Lower precision is however often traded for tractability.

8 Conclusion

Based on relevant literature (Amir and Chang, 2008; Yang et al., 2007; Balduccini, 2007; Certicky, 2012; Mourão et al., 2010; Zettlemoyer et al., 2005), we have identified a common collection of challenges, that the current action learning methods try to overcome. Each of these methods is able to deal with a different subset of these subproblems, which makes it applicable in different situations and domains. The relation between these challenges and the real-world applications of action learning methods has been clarified.

References

Amir, E. and Chang, A. (2008). Learning partially observable deterministic action models. *Journal of Artificial Intelligence Research*, 33(1):349–402.

Balduccini, M. (2007). Learning action descriptions with a-prolog: Action language c. In *AAAI Spring Symposium: Logical Formalizations of Commonsense Reasoning*, pages 13–18.

Borodin, A. and El-Yaniv, R. (1998). *Online computation and competitive analysis*. Cambridge University Press, New York, NY, USA.

Certicky, M. (2012). Action learning with reactive answer set programming: Preliminary report. In *ICAS 2012, The Eighth International Conference on Autonomic and Autonomous Systems*, pages 107–111.

Eiter, T., Faber, W., Leone, N., Pfeifer, G., and Polleres, A. (2000). Planning under incomplete knowledge. In *Proceedings of the First International Conference on Computational Logic*, CL '00, pages 807–821, London, UK, UK. Springer-Verlag.

Fikes, R. E. and Nilsson, N. J. (1971). Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3-4):189–208.

Ginsberg, M. L. and Smith, D. E. (1988). Reasoning about action i: A possible worlds approach. *Artificial Intelligence*, 35(2):165 – 195.

Giunchiglia, E. and Lifschitz, V. (1998). An action language based on causal explanation: preliminary report. In *Proceedings of the fifteenth national/tenth conference on Artificial intelligence/Innovative applications of artificial intelligence*, AAAI '98/IAAI '98, pages 623–630, Menlo Park, CA, USA. American Association for Artificial Intelligence.

Gupta, N. and Nau, D. S. (1992). On the complexity of blocks-world planning. *Artificial Intelligence*, 56(2-3):223–254.

Hopcroft, J. E. (2007). *Introduction to Automata Theory, Languages, and Computation*. Pearson Addison Wesley, 3rd edition.

McDermott, D., Ghallab, M., Howe, A., Knoblock, C., Ram, A., Veloso, M., Weld, D., and Wilkins, D. (1998). Pddl - the planning domain definition language. *Annals of Physics*, 54(CVC TR-98-003):26.

Mourão, K., Petrick, R. P. A., and Steedman, M. (2010). Learning action effects in partially observable domains. In *Proceedings of the 2010 conference on ECAI 2010: 19th European Conference on Artificial Intelligence*, pages 973–974, Amsterdam, The Netherlands, The Netherlands. IOS Press.

Nilsson, N. J. (1982). *Principles of Artificial Intelligence*.

Pednault, E. P. D. (1989). Adl: exploring the middle ground between strips and the situation calculus. In *Proceedings of the first international conference on Principles of knowledge representation and reasoning*, pages 324–332, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Russell, S. J. and Norvig, P. (2003). *Artificial Intelligence: A Modern Approach*. Pearson Education, 2 edition.

Slaney, J. and Thibaux, S. (2001). Blocks world revisited. *Artificial Intelligence*, 125(1-2):119–153.

Yang, Q., Wu, K., and Jiang, Y. (2007). Learning action models from plan examples using weighted max-sat. *Artificial Intelligence*, 171(2-3):107–143.

Zettlemoyer, L. S., Pasula, H. M., and Kaelblin, L. P. (2005). Learning planning rules in noisy stochastic worlds. In *IN AAAI*, pages 911–918. AAAI Press.