ASP-based Perspective on Reactive Policies with Planning

Zeynep G. Saribatur
Institute of Logic and Computation, TU Wien, Austria

Abstract

Answer Set Programming (ASP) is a highly expressive paradigm for declarative problem solving. Its declarative language can also be used to formalize actions, planning, and agent policies, in an expressive setting, and to reason about them. An ASP-based view on describing policies that express a reactive behavior for an agent employs the notion of indistinguishable states, and combines components for describing reactivity such as target establishment and (online) planning (Saribatur and Eiter 2016). The representation allows one to analyze the flow of executing the given reactive policy, and lays foundations for verifying properties of policies.

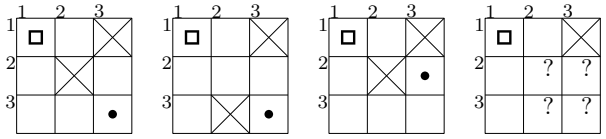
Reactive Behavior with Policies

In search scenarios, where an agent needs to find a missing person in unknown environments, the naive approach of searching for a plan that achieves the main goal easily becomes troublesome, as the planner needs to consider all possible initial states to find a plan that guarantees finding the person. Alternatively, a reactive policy can be described for the agent (e.g., "move to the farthest visible point") that determines its course of actions and guides the agent in the environment, while the agent gains information (e.g., obstacle locations) on the way. Then it needs to be checked whether the policy works or not, e.g., the person is always found, regardless of actual obstacle locations.

This reactive behavior can be seen as deciding the course of actions by determining targets as stepping stones to achieve during the interaction with the environment, where the agents come with an (online) planning capability to compute plans to reach the targets. Following the policy, the agent would determine a target at its current state, compute a plan to reach the target, execute it, observe the outcomes.

Fig. 1(a)-1(c) show some instances to demonstrate that the given policy might not always work. The farthest (horizontally/vertically) visible point in these states is (3,1), which is determined as the target. Then the policy computes a plan to reach this target. While in Fig. 1(a) the person will be found after executing the plan, in Fig. 1(b) following the policy will result in a loop and in Fig. 1(c) there is a possibility to loop, depending on the chosen target.

Figure 1: (a),(b),(c): Possible instances of a search scenario in a grid-cell environment, (d): Agent's observation in the instances, □:agent, •:person, ×:obstacle, ?:unknown



Answer Set Programming

Answer Set Programming (ASP) (Lifschitz 2008; Brewka *et al.* 2011; Baral 2003; Erdem *et al.* 2016) is a widely used problem solving approach, as its underlying non-monotonic semantics is general enough to represent interesting computational problems. Dynamic domains can be described by a "history program" (Lifschitz 1999), where the answer sets represent possible evolutions of the system over a time interval. This is achieved by adding a time variable to the atoms, and introducing action atoms that may cause changes over time. The expressiveness allows to describe the actions in a declarative way through their preconditions and direct/indirect effects, while addressing the problems encountered when reasoning about actions, e.g., the frame problem (McCarthy and Hayes 1969).

For illustration, the following rule describes a *direct effect* of the action $goTo$ over the robot's location.

$$rAt(X,Y,T{+}1) \leftarrow goTo(X,Y,T).$$

Actions can also have *indirect effects* over the state (rules not mentioning actions); e.g., the robot location is visited:

$$visited(X,Y,T) \leftarrow rAt(X,Y,T).$$

Inertia laws (unaffectedness) can be elegantly expressed, e.g., $rAt(X,Y,T{+}1) \leftarrow rAt(X,Y,T), not\ \neg rAt(X,Y,T{+}1).$ says that the robot location remains by default the same.

One can also give further restrictions on the state, e.g., the robot and an obstacle can never be in the same cell.

$$\bot \leftarrow rAt(X,Y,T), obsAt(X,Y,T).$$

Constraints can also define *preconditions* of an action, e.g.,

$$\bot \leftarrow goTo(X,Y,T), obsAt(X,Y,T).$$

Dedicated action languages carry this idea further with special syntax for such axioms (Gelfond and Lifschitz

1998), and can be translated to ASP (Giunchiglia *et al.* 2004). The expressiveness also allows to distinguish how the objects are affected by the actions. E.g., the obstacles are not affected by *goTo*, while they play a role in its executability.

Using ASP, the policies can be phrased in logical formulas which are understandable and easily changeable. E.g., target determination according to the policy is as below.

$$1\,\{target(X1,Y1,T) : farthest(X,Y,X1,Y1,T)\}\,1 \leftarrow rAt(X,Y,T), not\ seen(T), not\ caught(T).$$
$$target(X,Y,T) \leftarrow seen(T), not\ caught(T), pAt(X,Y,T).$$

After computing a target for a state, an outsourced planner can be used to determine the course of actions, i.e., *policy action*, from the agent's current location to the target location. Checking whether the policy works can be done by searching for a path following such policy actions, where the main goal is not reached. If such a path can not be found, then the policy works; otherwise, a counterexample to the policy's expected behavior is found.

Equalized States and Policy Actions

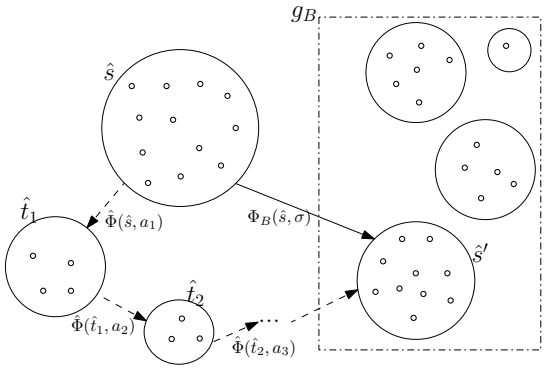
Depending on the agent's designed behavior, and its determination of its course of actions at a state, some information in the state may not be necessary, relevant or even observable. In this sense, the states that contain different facts about such information can be seen as *indistinguishable* to the agent. E.g., Fig. 1(a)-1(c) provide the same observations (Fig. 1(d)). Since the rest of the environment can not be observed, these states are indistinguishable to the agent.

The notion of equalized states is about clustering the states that contain the same *profile* according to the policy. E.g., for partially observable environments, same observations could yield the same profile. Such a clustering according to the observations is similarly considered in (Bonet and Geffner 2015). For fully observable environments, one needs to check the policy to determine the profiles.

The policy uses a target function to determine a target and an outsourced planner to compute a plan to reach the target. The policy actions are defined to be these plans that are determined according to the policy and executed. Such actions can also be seen as macro actions/big jumps, and the transitions between the equalized states are defined using them. The outsourced planner is considered to compute a conformant plan to the target, for the computed plans to be sound and complete. This notion is similar to the related work (Banihashemi *et al.* 2017) which considers a refinement mapping between a high-level action and low-level actions and defines the properties for such high-level view.

Fig. 2 demonstrates a policy action transition. Depending on the current state, $\hat{s}$, a plan σ can be executed if it is returned by the outsourced planner to reach the target g_B that is determined by the policy. There may be more than one equalized state satisfying g_B, and the policy execution function $\Phi_B(\hat{s},\sigma)$ executes σ and finds a transition into one of these states, $\hat{s}'$. When the knowledge of the executed plans are projected away to only consider $\Phi_B : \widehat{S} \rightarrow 2^{\widehat{S}}$, the transition Φ_B becomes a big jump between states, where the actions taken and the states passed in between are omitted.

Figure 2: A transition in the equalized transition system



A properness condition over the equalization ensures that having a trajectory through the policy actions guarantees the existence of a concrete trajectory. Knowing that any such trajectory found in the equalized transition system exists in the original transition system is enough to conclude that the policy indeed does not work.

Next Challenge. When such a macro action view may not help enough to the state explosion, an abstraction over the states and the policy actions by deliberately losing information becomes necessary.

References

Bita Banihashemi, Giuseppe De Giacomo, and Yves Lespérance. Abstraction in situation calculus action theories. In *Proc. AAAI*, pages 1048–1055, 2017.

Chitta Baral. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge Uni. Press, 2003.

Blai Bonet and Hector Geffner. Policies that generalize: Solving many planning problems with the same policy. In *Proc. IJCAI*, 2015.

Gerhard Brewka, Thomas Eiter, and Mirosaw Truszczyski. Answer set programming at a glance. *Communications of the ACM*, 54(12):92–103, 2011.

Esra Erdem, Michael Gelfond, and Nicola Leone. Applications of answer set programming. *AI Magazine*, 37(3):53–68, 2016.

Michael Gelfond and Vladimir Lifschitz. Action languages. *Electronic Transactions on AI*, 3(16), 1998.

Enrico Giunchiglia, Joohyung Lee, Vladimir Lifschitz, Norman McCain, and Hudson Turner. Nonmonotonic causal theories. *AIJ*, 153(1):49–104, 2004.

Vladimir Lifschitz. Answer set planning. In *Proc. ICLP*, pages 23–37, 1999.

Vladimir Lifschitz. What is answer set programming? In *Proc. AAAI*, pages 1594–1597, 2008.

John McCarthy and Patrick Hayes. *Some philosophical problems from the standpoint of artificial intelligence*. Stanford University USA, 1969.

Zeynep G. Saribatur and Thomas Eiter. Reactive policies with planning for action languages. In *Proc. JELIA*, volume 10021 of *LNCS*, pages 463–480. Springer, 2016.