Synthesis of infinite macro-actions

Departamento de Sistemas Informáticos y Computación
Universitat Politècnica de València.
Camino de Vera s/n. 46022 Valencia, Spain
{dieaigar,serjice,onaindia}@dsic.upv.es

Abstract

Introduction

A *macro-action* is a parameterized sequence of actions that has the form of a standard classical planning action so it can be reused straightforward to enrich a given planning domain theory. *Macro-planning* is a well-studied field and a wide number of macro planners exist, e.g., Marvin [Coles and Smith, 2007], MUM [Chrpa, 2010], MACRO FF [Boteaet al., 2005a], or DBMP/S [Hofmann et al., 2017] that leverage macro-actions to reduce the depth of the planning search space. All these existing macro-planners are restricted to *macro-actions* that represent a *finite and fixed* sequence of actions.

In this work we extend the notion of macros and define macro-actions that refer to a possibly *infinite* sequence of actions, then we show a representation of this kind of macro-actions using features with the form of *recursive derived predicates*. Last but not least, we present a classical planing compilation approach for the synthesis of *infinite macro-actions* from plans with an off-the-shelf classical planner.

To illustrate the notion of *infinite macros* Figure 1 shows `infinite-WALK`, an example of an *infinite macro-action* that is represented with the PDDL recursive derived predicate `infinite-path` and that encodes a possibly infinite sequence of `WALK` actions from the driverlog domain.

```
;;;
;;; Origininal action
;;;
(:action WALK
  :parameters (?d - driver ?from ?to - location)
  :precondition (and (at ?d ?from) (path ?from ?to))
  :effect (and (not (at ?d ?from)) (at ?d ?to)))


;;;
;;; Infinite macro-action
;;;

(:action infinite-WALK
  :parameters (?d - driver ?from ?to - location)
  :precondition (and (at ?d ?from) (infinite-path ?from ?to))
  :effect (and (not (at ?d ?from)) (at ?d ?to)))

(:derived infinite-path (?from ?to - location)
 (or (path ?from ?to)   ;;; base case
     (and (exists (?x - location) ;;; recursive case
                  (and (path ?from ?x) (infinite-path ?x ?to))))))
```

Figure 1: *Infinite macro-action* represented with the PDDL recursive derived predicate `infinite-path` and that encodes a possibly inifinite sequence of `WALK` actions from the driverlog domain.

Background

This section introduces the classical plannning model and the classical planning compilation for the learning of STRIPS actions.

Classical planning with conditional effects

F is the set of *fluents* or *state variables* (propositional variables). A *literal* l is a valuation of a fluent $f \in F$, i.e. either $l = f$ or $l = \neg f$. L is a set of literals that represents a partial assignment of values to fluents, and $\mathcal{L}(F)$ is the set of all literals sets on F, i.e. all partial assignments of values to fluents. A *state* s is a full assignment of values to fluents. We explicitly include negative literals $\neg f$ in states and so $|s| = |F|$ and the size of the state space is $2^{|F|}$.

A *planning frame* is a tuple $\Phi = \langle F, A\rangle$, where F is a set of fluents and A is a set of *actions*. An action $a \in A$ is defined with *preconditions*, $\mathsf{pre}(a) \in \mathcal{L}(F)$, and *effects* $\mathsf{eff}(a) \in \mathcal{L}(F)$. The semantics of actions $a \in A$ is specified with two functions: $\rho(s, a)$ denotes whether action a is *applicable* in a state s and $\theta(s, a)$ denotes the *successor state* that results of applying action a in a state s. Then, $\rho(s, a)$ holds iff $\mathsf{pre}(a) \subseteq s$. And the result of applying a in s is $\theta(s, a) = \{s \setminus \neg\mathsf{eff}(a)) \cup \mathsf{eff}(a)\}$, with $\neg\mathsf{eff}(a) = \{\neg l : l \in \mathsf{eff}(a)\}$.

A *planning problem* is defined as a tuple $P = \langle F, A, I, G\rangle$, where I is the initial state in which all the fluents of F are assigned a value true/false and G is the goal set. A *plan* π for P is an action sequence $\pi = \langle a_1, \dots, a_n\rangle$, and $|\pi| = n$ denotes its *plan length*. The execution of π in the initial state I of P induces a *trajectory* $\tau(\pi, P) = \langle s_0, a_1, s_1, \dots, a_n, s_n\rangle$ such that $s_0 = I$ and, for each $1 \leq i \leq n$, it holds $\rho(s_{i-1}, a_i)$ and $s_i = \theta(s_{i-1}, a_i)$. The *trajectory length* of $\tau(\pi, P)$ is given by the plan length of π. A trajectory $\tau(\pi, P)$ that solves P is one in which $G \subseteq s_n$.

An action $a_c \in A$ with conditional effects is defined as a set of preconditions $\mathsf{pre}(a_c) \in \mathcal{L}(F)$ and a set of *conditional effects* $\mathsf{cond}(a_c)$. Each conditional effect $C \rhd E \in \mathsf{cond}(a_c)$ is composed of two sets of literals: $C \in \mathcal{L}(F)$, the *condition*, and $E \in \mathcal{L}(F)$, the *effect*. An action $a_c \in A$ is applicable in a state s if and only if $\mathsf{pre}(a_c) \subseteq s$, and the *triggered effects* resulting from the action application are the effects whose conditions hold in s: $triggered(s, a_c) = \bigcup_{C \rhd E \in \mathsf{cond}(a_c), C \subseteq s} E$.

Learning action models as planning

The approach for learning STRIPS action models presented in (Aineto, Jiménez, and Onaindia 2018), which we will use as our baseline learning system (hereafter BLS, for short), is a compilation scheme that transforms the problem of learning the preconditions and effects of action models into a planning task P'. A STRIPS *action model* ξ is defined as $\xi = \langle name(\xi), pars(\xi), pre(\xi), add(\xi), del(\xi)\rangle$, where $name(\xi)$ and parameters, $pars(\xi)$, define the header of ξ; and $pre(\xi)$, $del(\xi)$ and $add(\xi)$) are sets of fluents that represent the *preconditions*, *negative effects* and *positive effects*, respectively, of the actions induced from the action model ξ.

The BLS receives as input an empty domain model, which only contains the headers of the action models, and a set of observations of plan executions, and creates a propositional encoding of the planning task P'. Let Ψ be the set of *predicates*[1] that shape the variables F. The set of propositions of P' that can appear in $pre(\xi)$, $del(\xi)$ and $add(\xi)$ of a given ξ, denoted as $\mathcal{I}_{\xi,\Psi}$, are FOL interpretations of Ψ over the parameters $pars(\xi)$. For instance, in a four-operator *blocksworld* (Slaney and Thiébaux 2001), the $\mathcal{I}_{\xi,\Psi}$ set contains five elements for the `pickup(`v_1`)` model, $\mathcal{I}_{pickup,\Psi}$={`handempty`, `holding(`$v_1$`)`, `clear(`$v_1$`)`, `ontable(`$v_1$`)`, `on(`$v_1,v_1$`)`} and eleven elements for the model of `stack(`v_1,v_2`)`, $\mathcal{I}_{stack,\Psi}$={`handempty`, `holding(`v_1`)`, `holding(`v_2`)`, `clear(`v_1`)`, `clear(`v_2`)`, `ontable(`v_1`)`, `ontable(`v_2`)`, `on(`v_1,v_1`)`, `on(`v_1,v_2`)`, `on(`v_2,v_1`)`, `on(`v_2,v_2`)`}. Hence, solving P' consists in determining which elements of $\mathcal{I}_{\xi,\Psi}$ will shape the preconditions, positive and negative effects of each action model ξ.

The decision as to whether or not an element of $\mathcal{I}_{\xi,\Psi}$ will be part of $pre(\xi)$, $del(\xi)$ or $add(\xi)$ is given by the plan that solves P'. Specifically, two different sets of actions are included in the definition of P': *insert actions*, which insert preconditions and effects on an action model; and *apply actions*, which validate the application of the learned action models in the input observations. Roughly speaking, in the *blocksworld* domain, the insert actions of a plan that solves P' will look like `(insert_pre_stack_holding_v1)`, `(insert_eff_stack_clear_v1)`, `(insert_eff_stack_clear_v2)`, where the second action denotes a positive effect and the third one a negative effect both to be inserted in the model of `stack`; and the second set of actions of the plan that solves P' will be like `(apply_unstack blockB blockA)`, `(validate_1)`, `(apply_putdown blockB)`, `(validate_2)`, where the `validate` actions denote the points at which the states generated through the `apply` actions must be validated with the observations of plan executions.

In a nutshell, the output of the BLS compilation is a plan that completes the empty input domain model by specifying the preconditions and effects of each action model such that the validation of the completed model over the input observations is successful.

[1] The initial state of an observation is a full assignment of values to fluents, $|s_0| = |F|$, and so the predicates Ψ are extractable from the observed state s_0.

```
(:action WALK-baseCase
  :parameters (?d - driver ?from ?x ?to - location)
  :precondition (and (at ?d ?x) (evaluating-path ?from ?to) (path ?x ?to))
  :effect (and (not (at ?d ?x)) (at ?d ?to)
               (infinite-path ?from ?to)
               (not (evaluating-path ?from ?to))))


(:action WALK-recursiveCase
  :parameters (?d - driver ?from ?x ?to - location)
  :precondition (and (evaluating-path ?from ?x) (at ?d ?x) (path ?from ?x))
  :effect (and (not (at ?d ?from)) (at ?d ?x)
               (not (evaluating-path ?from ?x))
               (evaluating-path ?from ?to)))
```

Figure 2: Two STRIPS actions representing a macro-action that encodes possibly inifinite sequence of WALK ations.

```
  actions for programming the precs/effs of the strips action WALK-baseCase
  actions for programming the precs/effs of WALK-recursiveCase
  (apply-WALK-recursiveCase d1 l1 l1 l2)
  (apply-WALK-recursiveCase d1 l1 l2 l3)
  (apply-WALK-recursiveCase d1 l1 l3 l4)
  (apply-WALK-baseCase d1 l1 l4 l5)
```

Figure 3: Sequential plan for learning the *macro-action* that encodes possibly inifinite sequence of WALK ations.

Synthesis of infinite macro-actions

Synthesis of infinite macro-actions with classical planning

Our approach is leveraging the classical planning compilation for the learning of strips actions to learn the preconditions and effects of two actions, one that represents the *case base* and another one that represents the *recursive case* of a recursively defined predicate (Aineto, Jiménez, and Onaindia 2018). Figure 2 shows the two STRIPS actions that represent a macro-action encoding a possibly inifinite sequence of WALK ations from the driverlog domain.

Figure 3 shows a sequential plan for sythesizing the *infinite macro-action* shown Figure 2. The plan is sythesized in a graph of locations that corresponds to a five-nodes linked list.

Acknowledgments

This work is supported by the Spanish MINECO project TIN2017-88476-C2-1-R. D. Aineto is partially supported by the *FPU16/03184* and S. Jiménez by the *RYC15/18009*. M. Ramírez research is partially funded by DST Group Joint & Operations Analysis Division.

References

Aineto, D.; Jiménez, S.; and Onaindia, E. 2018. Learning STRIPS action models with classical planning. In *International Conference on Automated Planning and Scheduling, (ICAPS-18)*, 399–407. AAAI Press.

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