Decompositional Planning PDDL Extension

Version 0.1

Rogelio E. Cardona-Rivera, Camille Barot

August 23, 2016

Abstract

This manual describes the extension to PDDL syntax needed to represent decompositional planning constructs. Decompositional planning is a planning model that combines hierarchical reasoning as discussed in hierarchical task networks and least-commitment refinement reasoning as discussed in partial-order causal link planning in a unified knowledge representation, and a sound and primitive complete reasoning procedure.

1 Introduction

This manual describes the syntax and discusses the semantics of an extension to the Planning Domain Definition Language needed to encode decompositional planning constructs. These constructs are centered on representing action decomposition as discussed in hierarchical task networks.

2 A Simple Example

We begin by an example of the decompositional planning extension in use. The example is listed in Figure 1, and represents a travel domain that describes an *abstract action* of traveling from one place to another, and a *decomposition schema* that refines the abstract action in terms of (other defined) concrete ones that accomplish it.

```
(define (domain travel)
  (:requirements :strips :decompositions)
  (:types  plane car - transport
           person place transport - object)
  (:predicates
      (in  ?p - person ?t - transport)
      (at  ?t - transport ?p - place)
      (at  ?pe - person ?pl - place))

  (:action travel
      :parameters   (?p - person ?f - place ?to - place ?tr - transport)
      :precondition (at ?p ?f)
      :effect       (and (not (at ?p ?f)) (at ?p ?t))
      :composite    t)

  (:action get-in-car
      :parameters   (?pe - person ?c - car ?pl - place)
      :precondition (and (at ?pe ?pl) (at ?c ?pl))
      :effect       (and (not (at ?pe ?pl)) (in ?pe ?c))
      :composite    f)

  (:action drive
      :parameters   (?p - person ?c - car ?f - place ?t - place)
      :precondition (and (at ?c ?f) (in ?p ?c))
      :effect       (and (not (at ?c ?f)) (at ?c ?t)))

  (:action get-out-of-car
      :parameters   (?pe - person ?c - car ?pl - place)
      :precondition (and (at ?c ?pl) (in ?pe ?c))
      :effect       (and (not (in ?pe ?c)) (at ?pe ?pl)))

  (:decomposition travel
      :name drive
      :parameters (?p - person ?f - place ?t - place ?c - car)
      :steps      ( (step1 (get-in-car ?p ?c ?t))
                    (step2 (drive ?p ?c ?f ?t))
                    (step3 (get-out-of-car ?p ?c ?f)))
      :links      ( (step1 (in ?p ?c) step2)
                    (step1 (in ?p ?c) step3)
                    (step2 (at ?c ?t) step3) )))
```

Figure 1: An example decompositional domain definition. This listing encodes actions in a travel domain.

3 Requirements flags

Requirement	*Description*
`:decomposition`	Allows action to have associated `:decomposition` operators

4 Composite actions and Decomposition schemata

In this section, we present our notation for encoding composite actions and decomposition schemata using an Extended Backus-Naur Form (EBNF). We borrow the definitions and conventions outlined in version 1.2 of the Planning Domain Definition Language [McDermott et al., 1998].

If an action is composite, it must be marked explicitly as being non-primitive within the action's definition:

```
<action-def body>  ::=  Definitions as in PDDL1.2 [McDermott et al., 1998]
                        [:composite <boolean>]
```

When an action is marked as composite, it means that it is not directly executable in a planning domain. In order to execute it, an appropriate decomposition must be specified. Decompositions are defined outside of actions:

```
<decomposition-def>  ::= (:decomposition <action functor>
                          <decomposition-body>)
```

A decomposition contains a subplan, consisting of a set of steps, causal links, and temporal orderings.

```
<decomposition-body>  ::= [:name <name>]
                          :parameters (<typed list (variable)>)
                          :steps (<step>*)
                          [:links (<link>*)]
                          [:orderings (<ordering>*)]
<step>                ::=:conditional-components
<step>                ::= (<name> (<action functor> <term>*))
<link>                ::= (<name> <condition> <name>)
<condition>           ::= <atomic formula(term)>
<condition>           ::= (not <atomic formula(term)>)
<ordering>            ::= (<name> <name>)
```

References

[McDermott et al., 1998] McDermott, D., Ghallab, M., Howe, A., Knoblock, C., Ram, A., Veloso, M., Weld, D., and Wilkins, D. (1998). Pddl-the planning domain definition language.