Generating Abstractions from Static Domain Analysis

Eloisa Vargiu
(joint work with Giuliano Armano and Giancarlo Cherchi)

Department of Electrical and Electronic Engineering
Univerisity of Cagliari
Outline of the Talk

- Introduction
- The Theoretical Perspective
- The Pragmatic Perspective
- Experiments with the DriverDepot Domain
- Conclusions and Future Work
Introduction

- How to implement a proactive behavior in an agent using abstraction techniques to deal with the complexity of the domains
- Why abstractions?
  - Theoretically abstraction must be used to render agents able to solve complex problems
  - Pragmatically we proposed a technique devised to generate abstract spaces starting from a “ground” description of the domain being modeled
The Theoretical Perspective
Hierarchical Planning

- An effective approach for dealing with the inherent complexity of agent tasks
- Exploits an ordered set of *abstractions* for controlling the search
- Under certain assumptions, it can reduce the size of the search space from exponential to linear in the size of the solution
Abstraction as Control Heuristics

- *Abstraction* is a technique aimed at providing some control heuristics (*).

- The original search space is mapped into corresponding abstract spaces, in which irrelevant details are disregarded at different level of granularity.

  (*) abstract levels are used to control the search at the ground level.
Hierarchical Planning at a Glance

... on a two-levels abstraction hierarchy
Abstraction: General Perspective

- Generally speaking, abstraction might occur on *predicates, types, and operators*.
- Typical examples of abstraction based on:
  - Predicates/Types → relaxed and reduced models
  - Operators → macro-operators
The Pragmatic Perspective
Using Abstraction

- What do we need to perform planning by abstraction?
  - An *engine* able to enforce hierarchical planning
    - HW[ ]
  - A *syntactic support* to express abstraction hierarchies (based on PDDL)
    - HPDDL
  - A *technique* to generate abstraction hierarchies
    - ABSGEN
HW[ ]... (I)

Problem hierarchy

Domain hierarchy

Engine

Solution

Embedded Planner

HW[P] = Hierarchical Wrapper

WOA 2003 Dagli Oggetti agli Agenti - Sistemi Intelligenti e Computazione Pervasiva
**HW[]...**

- **HW[P]** is an instance of **HW[ ]**
- where...
  - Embodies a generic planner $P$
  - Delegates to $P$ the search at any required level of abstraction
  - Performs a suitable switching between abstraction levels
To our knowledge, existing planning systems tailored for abstraction did not resort to a common notation.

To contrast this lack of a standardization, we have proposed an extension to PDDL able to represent *abstraction hierarchies*. 

**HPDDL...**

(I)
Defining an abstraction hierarchy using HPDDL:

(define (hierarchy <name>))
 (:domains <domain-name>+)
 (:mapping (<src-domain> <dst-domain>)
  [:types <types-def>]
  [:predicates <predicates-def>]
  [:actions <actions-def>])*)
Abstraction hierarchies can be hand-coded by a domain engineer. To facilitate the setting of the abstraction hierarchies, a novel semi-automatic technique has been devised.
Involves executing two steps:

- (step 1) searching for macro-operator \textit{(fully automated)}
- (step 2) selecting some relevant macro-operators and translating them into abstract operators \textit{(semi-automated)}
Step 1 is performed by:
- 1a) building and
- 2a) pruning a directed graph (*)

(*) where:
- nodes represent operators, and
- edges represent relations between effects of the source node and preconditions of the destination node
1a) building the graph

\[\text{OP1} \quad \langle 1 \mid 1 \rangle \quad \text{OP2}\]

\[
\text{[...] :effects}
\]
\[
(p1 \ ?o1 \ ?o2)
(p2 \ ?o2 \ ?o3)
(not \ (p3 \ ?o4))
\]

\[
\text{[...] :preconditions}
\]
\[
(p1 \ ?o1 \ ?o2)
(p3 \ ?o4)
\]
1b) pruning the graph

It is performed according to some domain-independent **heuristics**.
Step2 (*) is performed by:

2a) selecting a subset of the resulting macro-operators, and
2b) translating them into abstract operators

(*) not completely automated
2a) selecting a subset of the resulting macro-operators...

- the most promising macro-operators can be easily extracted from the pruned graph, each path being related with a candidate macro-operator
- sequences that do not alter the state of the world are not considered meaningful
2b) translating the chosen macro-operators into abstract operators (*)...

- the simplest way consists of deleting from the abstract level all predicates that do not occur among preconditions or effects of any selected macro-operator
- this process influences (and is influenced by) the translation rules that apply to both types and predicates

(*) This is not a deterministic task
Experiments with the DriverDepot Domain
The Ground Domain (I)
The Ground Domain (II)

(define (domain DriverDepot-ground)
  [...]
  (:types
   place locatable - object
deport distributor - place
driver truck hoist surface - locatable
pallet crate - surface)
  (:predicates
   (available ?h - hoist) (clear ?s - surface)
   (driving ?d - driver ?t - truck) (empty ?t - truck))
  (:action Drive [... ])
  (:action Lift [... ])
  (:action Drop [... ])
  (:action Load [... ])
  (:action Unload [... ])
  (:action Board [... ])
  (:action Disembark [... ])
  (:action Walk [... ]))
The Corresponding Graph

- 'lift'
- 'load'
- 'drop'
- 'unload'
- 'drive'
- 'board'
- 'disembark'
- 'walk'

The diagram shows directed edges between the nodes with labels '<1 0>' or '<0 1>'.
The Pruned Graph

- lift
  - drop
  - load
  - unload
- drive
- board
  - disembark
  - walk
The Corresponding Macro-operators
The Selected Macro-Operator Schemata

<table>
<thead>
<tr>
<th>Macro-Operator Schema</th>
<th>Ground Sequence</th>
<th>Preconditions</th>
<th>Effects</th>
</tr>
</thead>
<tbody>
<tr>
<td>?h - hoist ?t - truck</td>
<td>drop</td>
<td>(available ?h)</td>
<td></td>
</tr>
<tr>
<td>?p1 ?p2 - place</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>?d - driver</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>?c - crate ?s - surface</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>?h - hoist ?t - truck</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>?p - place ?d - driver</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>?c - crate ?s - surface</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>?d - driver</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>?p1 ?p2 place ?t - truck</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>?d - driver</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>?t - truck</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>?p1 ?p2 - place</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>?h - hoist ?c - crate</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>?t - truck</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>?s - surface ?p - place</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Abstractions on Types

- object
  - place
    - depot
    - distributor
  - locatable
    - surface
    - hoist
    - truck
    - driver
    - crate
    - pallet
The Abstract Domain (I)

place1

crate1 pallet1 crate2 crate3

place2

crate4 driver truck1
(define (domain DriverDepot-abstract)
  
  (:types
    place locatable - object
    driver truck surface - locatable
    pallet crate - surface)

  (:predicates
    (at ?l - locatable ?p - place)
    (on ?c - crate ?s - surface)
    (in ?c - crate ?t - truck)
    (clear ?s - surface)
    (driving ?d - driver ?t - truck)
    (empty ?t - truck))

  (:action DriveUnloadDrop [...])
  (:action LiftLoad [...])
  (:action WalkBoard [...])
  (:action DriveDisembark [...])
  (:action Drive [...])
  )
Conclusions

- Abstraction must be used to render agents able to solve complex problems.
- A technique devised to generate abstract spaces starting from a “ground” description of the domain being modeled has been described.
Future Work

- Generating abstract spaces in a fully automated way
- Experimenting the system on more complex domains
- Extending the technique for dealing with PDDL 2.1 specifications