PDDL Domain Repair: Fixing Domains with Incomplete Action Effects

Alba Gragera - Raquel Fuentetaja - Ángel García-Olaya - Fernando Fernández

Computer Science and Engineering Department, Universidad Carlos III de Madrid, Spain

Motivation

- Automated planning typically assumes an accurate task specification, defined by a domain and a problem description, specified in PDDL.
- Modeling a planning task is complex and error-prone, requiring broad knowledge of the domain, the current task, and the formal language.
- An incomplete model may render the planning task unsolvable.

Explaining the absence of a solution in such cases is essential to support humans in the development of automated planning tasks.

Approach to repair flawed domains

The unsolvable task is compiled into a new planning task where:

- Actions can be repaired to insert possible missing effects.
- The solution is a plan that achieves the goals of the original problem while at the same time repairs the original actions.

Input Example

A Blocksworld planning domain with the following missing effects:

- **holding** effect from the **pick-up** action
- **on** effect from the **stack** action
- **holding** effect from the **unstack** action

```
(action pick-up
 :parameters (?x - block)
 :precondition (and (clear ?x) (ontable ?x) (handempty))
 :effect (and (not (ontable ?x))(not (clear ?x))(not (handempty))))
```

Classical Planning Compilation

- New predicates to represent the planning task elements
- Control predicates to manage the repair process
- New actions to modify the original actions: fix, add-fix, del-fix, close
- Bias to guide the repair using action costs

```
(action fix___adding_diff
 :parameters (?y - predicate ?a - action)
 :precondition (and (current-action ?a) (?a ?y))
 :effect (and (not ?a))(not (not ?a)))
```

Output Example

The result of the compiled planning task is a plan that includes the reparations made in the domain to achieve the goals.

```
(unstack b4 b3)
(fix___adding_diff holding unstack)
(add-fix____1par holding unstack b4 t_block)
(completed_fixed unstack)
(put-down b4)
(completed_no_fixed put-down)
(unstack b3 b2)
(add-fix____1par holding unstack b3 t_block)
(completed_fixed unstack)
(stack b3 b4)
(fix___adding_diff on stack)
(add-fix____2par_goal on stack b3 b4 t_block t_block)
(completed_fixed stack)
(pick-up b1)
(fix___adding_diff holding pick-up)
(add-fix____1par holding pick-up b1 t_block)
(completed_fixed pick-up)
(stack b1 b3)
(add-fix____2par_goal on stack b1 b3 t_block t_block)
(completed_fixed stack)
```

The plan is parsed and shown in the interface as repair suggestions to the original domain...

Conclusions

- A fairly accurate reparation without requiring additional information from the user, only a domain and a single problem.
- The lack of information about the number and location of flaws, as well as the user’s mental model, can lead to estimated repairs.