A Visual Studio Code Extension for Automatically Repairing Planning Domains

Songtuan Lin[*], **Mohammad Yousfi**[*], **Pascal Bercher**

School of Computing, The Australian National University
{firstName.lastName}@anu.edu.au

Abstract

We demonstrate a Visual Studio Code extension which aims at providing modeling assistance for modeling planning domains in PDDL, which serves as a front-end of our previous work. The extension can identify potential flaws in a domain and propose respective corrections by taking as input a set of counter-example plans, which are known to be valid but actually contradict the domain. Those input plans shall be provided by the user. The flaws are then identified and corrected by making changes to the domain so as to turn those plans into solutions, i.e., the changes are regarded as potential corrections to the domain. The extension supports corrections that add predicates to or remove predicates from actions' preconditions and effects.

Introduction

The complexity of modeling a planning domain has emerged as a major obstacle for deploying planning techniques more broadly, raising the demand for tools for providing modeling assistance. To contribute toward this direction, we present a plugin for Visual Studio Code which can identify and correct potential errors in a domain in PDDL (see the work by Haslum et al. (2019) for an introduction to PDDL).

The plugin serves as the front-end of our earlier technique (Lin, Grastien, and Bercher 2023). More concretely, the plugin takes as input a set of counter-example plans which are supposed to be solutions but are actually not, due to errors in the domain. It then proposes corrections to the user which are changes to the domain that turn the input plans into solutions. After receiving the proposed corrections, the user could justify which changes are desired and which are not. The justification will then be sent back to the plugin, starting another iteration of domain correction where the true corrections will be kept while the false changes will be forbidden. This process continues until the user is satisfied with all the changes proposed by the plugin.

Plugin Description

After presenting the general workflow of this plugin, we now describe in more detail how it works. Currently, our extension supports PDDL 1.2. The corrections our plugin can provide are restricted to adding predicates to and deleting predicates from actions' preconditions and effects, including both positive and negative ones, i.e., exactly those described by Lin, Grastien, and Bercher (2023).

[*]These authors contributed equally.

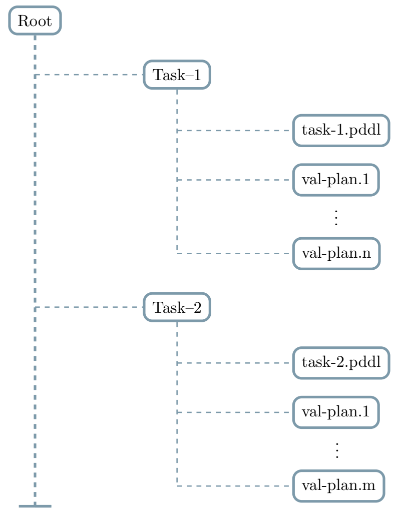


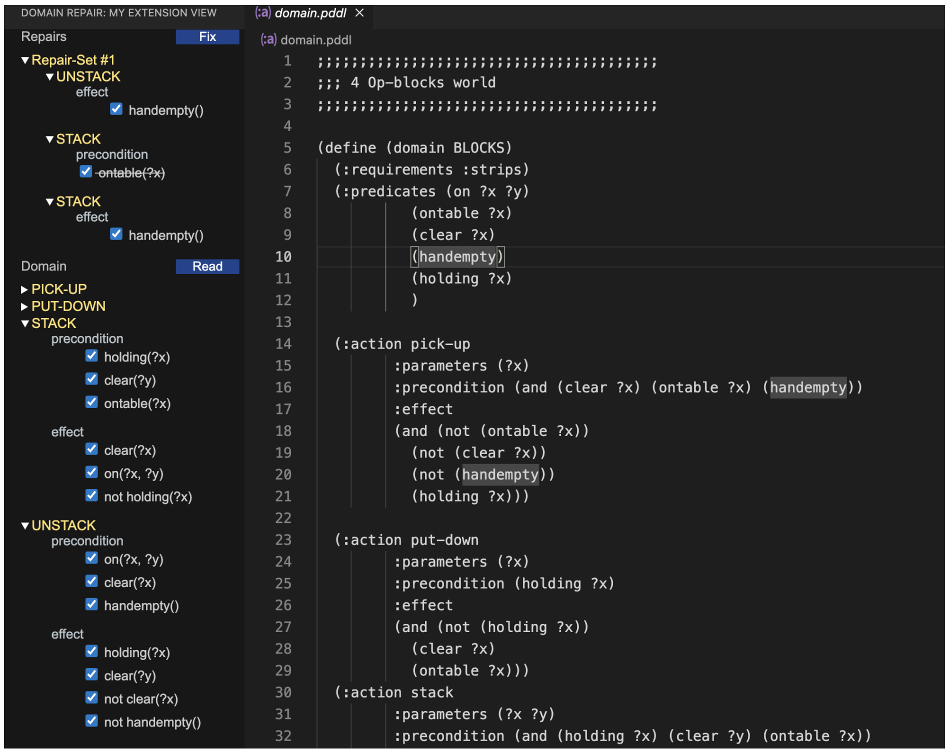
Figure 1: The structure of the folder containing all counter-example plans.

To use the extension, the user should first place the domain (in PDDL) to be corrected and a folder containing all counter-example plans into the same directory. In particular, the folder that contains the counter-example plans should be structured as follows:

1) There should be one subdirectory for *each* planning task with respect to the domain.
2) Each subdirectory with respect to a planning task should contain the respective task file (in PDDL) together with plan files each of which contains a plan that is supposed to be a solution to the planning problem defined by the domain and the planning task.
3) The name of a plan file should be `val-plan.i` where `i` is the index of the plan file in the directory with respect to a planning task.

An example of the structure of the folder is shown in Fig. 1.

After that, the user needs to open the directory containing both the domain and the folder of input plans in Visual Studio Code as a workplace. The next step is to open the domain file in the editor. Fig. 2 demonstrates the interface of the extension with an example input domain. The input domain



Repairs

- Repair-Set #1
  - UNSTACK
    - effect
      - ✓ handempty()
  - STACK
    - precondition
      - ✓ ~~ontable(?x)~~
  - STACK
    - effect
      - ✓ handempty()

Domain

- PICK-UP
- PUT-DOWN
- STACK
  - precondition
    - ✓ holding(?x)
    - ✓ clear(?y)
    - ✓ ontable(?x)
  - effect
    - ✓ clear(?x)
    - ✓ on(?x, ?y)
    - ✓ not holding(?x)
- UNSTACK
  - precondition
    - ✓ on(?x, ?y)
    - ✓ clear(?x)
    - ✓ handempty()
  - effect
    - ✓ holding(?x)
    - ✓ clear(?y)
    - ✓ not clear(?x)
    - ✓ not handempty()

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; 4 Op-blocks world
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(define (domain BLOCKS)
  (:requirements :strips)
  (:predicates (on ?x ?y)
          (ontable ?x)
          (clear ?x)
          (handempty)
          (holding ?x)
          )

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

  (:action put-down
        :parameters (?x)
        :precondition (holding ?x)
        :effect
        (and (not (holding ?x))
          (clear ?x)
          (ontable ?x)))
  (:action stack
        :parameters (?x ?y)
        :precondition (and (holding ?x) (clear ?y) (ontable ?x))
```

Figure 2: The user interface of the plugin. The upper part of the left panel shows the history of proposed corrections on each iteration while the lower part displays the domain. If a checkbox is unmarked, it means that the corresponding component is unmodifiable, i.e., the respective correction is forbidden.

is the BlocksWorld domain with the following errors: The predicate `handempty` is missing from the positive effects of the actions `stack` and `unstack`, and the precondition of `stack` has an extra predicate `ontable(?x)`. Once the domain file is opened, the user could click the “Read” button on the lower part of the left panel. This will let the extension read the domain file and display it (see the lower part of the left panel in Fig. 2). If a checkbox in the displayed domain is *unmarked*, it means that the respective component is *unchangeable*. To start correcting the domain, the user could click the “Fix” button. The extension will then return the proposed corrections, under the label “Repair Set” followed by the index of the current iteration A correction which is crossed out indicates that the respective predicate should be *removed* from the corresponding component of the action, e.g., the second correction displayed in Fig. 2. A correction that is *not* crossed out means that the predicate is added, e.g., the first and the third correction in Fig. 2. The user could *unmark* the checkbox associated with a correction to indicate that this correction is undesired. The next iteration will start with those corrections being forbidden.

Discussion and Conclusion

In this paper, we present a VS code extension for correcting errors in a domain by turning a set of counter-example plans into solutions. Note that there exists another tool (Gragera et al. 2023b,a) which was developed for the same purpose as ours, i.e., fixing errors in a domain. However, their underlying approach is different from ours which finds corrections to a domain by turning an unsolvable planning problem into a solvable one while allowing only adding positive effects to actions. One remark is that our extension can also make unsolvable problems solvable if plan traces are provided.

Acknowledgements

Pascal Bercher is the recipient of an Australian Research Council (ARC) Discovery Early Career Researcher Award (DECRA), project number DE240101245, funded by the Australian Government.

References

Gragera, A.; Fuentetaja, R.; Olaya, Á. G.; and Fernández, F. 2023a. PDDL Domain Repair: Fixing Domains with Incomplete Action Effects. In *ICAPS–Demo 2023*.

Gragera, A.; Fuentetaja, R.; Olaya, Á. G.; and Fernández, F. 2023b. A Planning Approach to Repair Domains with Incomplete Action Effects. In *ICAPS 2023*, 153–161. AAAI.

Haslum, P.; Muise, C.; Magazzeni, D.; and Lipovetzky, N. 2019. *An Introduction to the Planning Domain Definition Language*. Springer.

Lin, S.; Grastien, A.; and Bercher, P. 2023. Towards Automated Modeling Assistance: An Efficient Approach for Repairing Flawed Planning Domains. In *AAAI 2023*, 12022–12031. AAAI.