Detecting AI Planning Modelling Mistakes – Potential Errors and Benchmark Domains

Kayleigh Sleath and Pascal Bercher

School of Computing
The Australian National University
`firstName.lastName@anu.edu.au`

Abstract. AI planning systems can solve complex problems, leaving domain creation as one of the largest obstacles to a large-scale application of this technology. Domain modeling is a tedious, error-prone and manual process. Unfortunately, domain modelling assistance software is sparse and mostly restricted to editors with only surface-level functionality such as syntax highlighting. We address this important gap by proposing a list of potential domain errors which can be detected by problem parsers and modeling tools. We test well-known planning systems and modeling editors on models with those errors and report their results.

Keywords: Automated Planning · Modelling support · Knowledge Engineering · PDDL Modeling · HDDL Modeling

1 Introduction

Automated planning, a branch of artificial intelligence (AI), is concerned with generating sequences of actions that turn one state of a system into a desired one. This requires a formal specification of the planning model, which is written in text files and adheres to a specific syntax such as the planning domain description language (PDDL) [3] for classical (non-hierarchical) planning and its extension HDDL [6] for hierarchical task network (HTN) planning [1].

Although modeling is a complex and error-prone task, the few existing modeling tools focus on syntax highlighting, integrating a planner, and visualizing solutions – but they are of limited use if the domain modeler makes mistakes. Planning systems' parsers provide even less support for detecting such errors; in many cases they just crash, or even worse they don't find solutions or find wrong or inconsistent ones. There are a few more evolved works for modeling support [7], but they all assume a syntactically correct model and are hence orthogonal to our contributions.

To address this problem, we make the following contributions: *(1)* We provide a compilation of potential modeling errors. *(2)* We supply a public repository of 56 (flawed) benchmark domains containing each of these errors, to the best of our knowledge the first benchmark database for AI modeling support. *(3)* We conduct an evaluation of well-known AI planning tools for their ability to diagnose those errors, showing that not a single tool is able to spot all errors, with no tool being strictly stronger than another.

2 AI Planning Formalism

Due to space restrictions we do not provide a formal introduction to the description languages PDDL [3] and HDDL [6] or their underlying formalisms [1] and only refer to the respective literature. Instead, we explain the input languages based on a PDDL example taken from the PDDL textbook [3].

Listing 1.1. A PDDL action for moving a truck between locations [3].

```
(:action drive
  :parameters (?t - truck ?from ?to - location)
  :precondition (at ?t ?from)
  :effect (and (not (at ?t ?from)) (at ?to)))
```

Classical planning evolves around the transition of states, finite set of facts, propositions that encode what's currently true. States are changed by actions (see Listing 1.1), which have preconditions (here that the truck is at the location **?from**) and effects, specifying how the respective state changes (here that the truck is not at **?from** anymore but at **?to**). Problems are defined in a "lifted" fashion, where variables (parameters) are used to abstract away from concrete constants such as specific trucks or locations. These constants/objects are given in the problem description so that actions can be instantiated as required.

Hierarchical planning adds further constraints [1]. Here, we are additionally given a set of compound tasks and a set of decomposition methods that specify how these tasks could be refined into more primitive tasks and finally into actions. This process is quite similar to formal grammars, where production rules (corresponding to decomposition methods) are used to turn non-terminal symbols (compounds tasks) into terminal symbols (actions). The goal is to turn an initially given task network – a partially ordered sequence of tasks – into an executable action sequence (just as in classical planning), but now tasks can only be obtained by adhering to the hierarchy defined by the decomposition methods.

3 Potential Errors in Planning Domains

This section details a list of errors or potential errors that may be encountered when modelling planning domains in PDDL/HDDL, separated into:

- syntax errors: these are actual errors, but often not spotted by parsers and
- semantics errors: these would be warnings as they indicate a *potential* modeling error, to be checked by the domain modeler.

All errors we identify for classical planning (PDDL) naturally transfer to HTN planning (HDDL) as well, whereas HTN errors are unique to HTN planning. We hence present the respective flaws in two different lists.

The list has been translated into a repository[1] which we see as a first step towards a public testbed for PDDL and HDDL parsers. *We invite others to add additional cases we might not have thought of.*

[1] https://github.com/ProfDrChaos/flawedPlanningModels

3.1 Syntax Errors

(1) Inconsistent Parameter Use. The modeller attempts to use a predicate or task with either a parameter of an incompatible type or a different number of parameters than it was defined with. This second error is only possible in HTN planning since in classical planning actions are only defined once (and thus never referenced anymore), whereas HTN planning could re-use a task (primitive or compound) multiple times in decomposition methods.

(2) Undefined Entities. The modeller attempts to use an undefined predicate, type, or task. (Again "using undefined tasks" is only possible in HTN planning for the same reason as mentioned above.)

(3) General Syntax Errors. The modeller forgets to include a key piece of syntax or makes a typo - for instance, forgets to write ":parameters" in a task definition (which lists the sequence of typed task parameters), adds an extra parenthesis, forgets a dash when defining a variable, or forgets to write a questionmark in front of a variable name to differentiate it from a constant. It is expected that most of these errors are captured by any parser, but not all are, and useful error messages are not always produced.

(4) Duplicated Definitions. The modeller repeats some definitions (e.g., some task, decomposition method, predicate, or constant). Closely related, the modeller writes duplicate entries in a task definition – for example, includes multiple ":parameters" entries.

(5) Cyclic Type Declaration. When two types are directly or indirectly declared to be subtypes of each other, forming a cycle.

(6) Undeclared Parameters. The modeller tries to use a variable in the definition of a task (or decomposition method in case of HTN planning) that wasn't declared as a parameter of that task (or method).

(7) Cyclic Ordering Constraints. Task networks are defined over a partial ordering – which excludes cycles. – **HTN-specific**

(8) Duplicate Orderings. A method contains both the "ordered subtasks" keyword (which implies that only a *sequence* of tasks is provided), but also a (thus redundant) set of explicit ordering constraints. – **HTN-specific**

3.2 Semantic Errors

These are *potential* errors, which do not contradict PDDL/HDDL.

(9) Complementary Effects. There is an intersection between the ground negated and positive effects of a task.

(10) Unsatisfied Preconditions. Some action's preconditions can never be fulfilled. This may be due to syntactically complementary preconditions (with identical predicates, including parameters), or simply since in the given planning problem the precondition can't be made true. While the first possible cause is a simple syntax check the second involves complex reasoning, which is as hard as planning.

(11) Unused Elements. The modeller defines a type or predicate or a parameter in a task (or decomposition method in case of HTN planning) that is not used. In the case of HTN planning, tasks may be "unused", which can be defined as being unreachable from the initial task network.

(12) Redundant Effects. Some effect will never change the state to which the respective action is applied. There are two possibilities how this can happen: The simplest case is if some effect also occurs as a precondition (with identical parameters). The redundancy can however also be problem-dependent, i.e., if any grounding of some effect is contained in any state in which the respective action is applicable.

(13) Immutable Predicate. A predicate is defined which never occurs in task effects. This means the state of that predicate is constant.

(14) Compound Tasks Without a Primitive Refinement. The modeller defines a compound task which can never be refined into a primitive plan (it is therefore useless). A special case of this is not providing any decomposition method for some compound task. – **HTN-specific**

4 Evaluation of Existing Parsers

From the proposed list, we created a large benchmark set with flawed domains. We tested some of the best-known AI planning tools (planning system parsers or domain editors) that parse PDDL and HDDL domains and evaluated their performance. These tools were: editor.planning.domains [9], Visual Studio PDDL Plugin [2], Fast Downward [4], PANDA [5], HyperTensioN [8], and LiloTane [10].

4.1 Results

The software was evaluated for each flawed domain based on three categories:

Error Detection. Whether the software recognises the error and stops the parsing process ('yes' – green), crashes without catching the error ('crashes' – yellow), or provides a solution/reports unsolvable despite the model being wrong ('no' – red).

Location Guidance. Whether the software pinpoints the correct line number of the error ('yes' – green), points toward the correct area of code, usually by naming the task which contains the error ('close' – yellow), or provides an inaccurate or no indication of the location of the error ('no' – red).

Error Description. Whether the software provides a clear and helpful description of the error ('yes' – green), a correct description which is unclear or confusing ('close' – yellow), or no or incorrect error description ('no' – red).

The results are reported in Fig. 1 for the individual domains, and in Fig. 2 with an overview. We also provide all data collected (including the actual output messages of the tested software) in a Zenodo repository [11]. We can report that none of the software tested addressed any of our potential semantics errors, with the exception of the VSCode plugin diagnosing the unused predicate error.

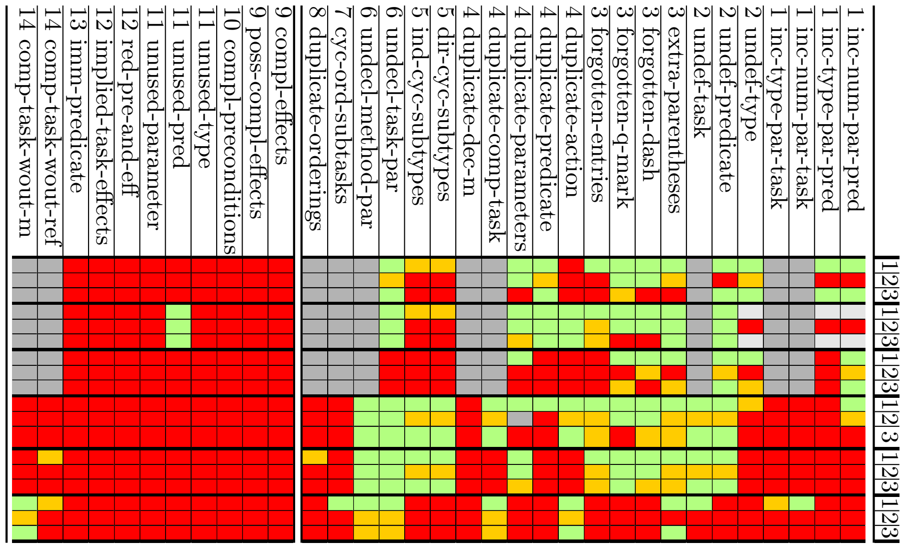

Fig. 1. Results of each software: Planning.Domains (PD), VSCode plugin (VS), Fast Downward (FD), (PAN)DA, Lilotane (LT), Hypertension (HT). We tested error detection (1), line pinpointing (2), and error message quality (3). We first list syntax errors, then (potential) semantic errors. For VS, the lighter shade of green corresponds to errors caught by PD, which the plugin uses as default planner.

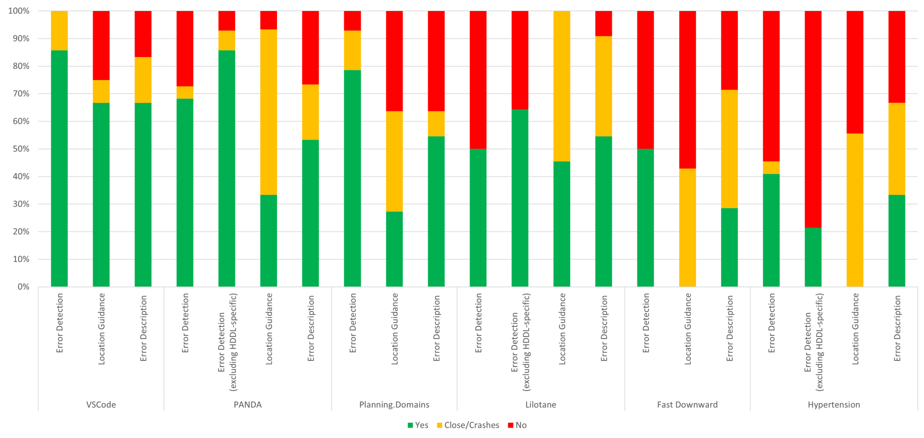

Fig. 2. The overall success rates of the evaluated software. The location guidance and error description rates are percentages of the number of errors caught by the parser, not the total number of errors tested (e.g. if a parser catches 6 of 10 errors, and provides a helpful error message for 3 of them, its success rate for Error Description would be 50%). For hierarchical planners, performance on only the domains which were tested on both classical and hierarchical planning systems is included (called 'excluding HDDL-specific') to allow for fair comparison between the two kinds.

5 Conclusion

We provided a comprehensive list of potential domain modelling errors for classical and hierarchical AI planning. It is accompanied by example domains containing each of these errors, proposed to form the foundation of a set of standardized tests for domain modelling assistance software and improving existing and future PDDL and HDDL parsers.

In our empirical evaluation, we show that a selection of successful well-known – and thus often used – AI planning systems and modeling tools for both PDDL and HDDL domains fail to recognize many of these errors. We thus hope that our list and benchmark set will act as a valuable contribution towards improving these and future software. We furthermore hope that other domain modelers see the benefit in our list and these test cases and thus provide additional benchmarks themselves.

Acknowledgements

We would like to thank Bernd Schattenberg for discussions (and one of the reported errors).

References

1. Bercher, P., Alford, R., Höller, D.: A survey on hierarchical planning - one abstract idea, many concrete realizations. In: Proc. of the 28th Int. Joint Conference on AI (IJCAI). pp. 6267–6275. IJCAI (2019)
2. Dolejsi, J.: PDDL visualstudio plugin (2017), https://marketplace.visualstudio.com/items? itemName=jan-dolejsi.pddl
3. Haslum, P., Lipovetzky, N., Magazzeni, D., Muise, C.: An Introduction to the Planning Domain Definition Language. Morgan & Claypool (2019)
4. Helmert, M.: The fast downward planning system. Journal of Artificial Intelligence Research (JAIR) **26**, 191–246 (2006)
5. Höller, D., Behnke, G., Bercher, P., Biundo, S.: The PANDA framework for hierarchical planning. Künstliche Intelligenz **35**(3), 391–396 (2021)
6. Höller, D., Behnke, G., Bercher, P., Biundo, S., Fiorino, H., Pellier, D., Alford, R.: HDDL: An extension to PDDL for expressing hierarchical planning problems. In: Proc. of the 34th AAAI Conf. on AI (AAAI). pp. 9883–9891. AAAI Press (2020)
7. Lin, S., Grastien, A., Bercher, P.: Towards automated modeling assistance: An efficient approach for repairing flawed planning domains. In: Proc. of the 37th AAAI Conference on AI (AAAI). pp. 12022–12031. AAAI Press (2023)
8. Magnaguagno, M.C., Meneguzzi, F., Silva, L.d.: HyperTensioN: A three-stage compiler for planning. In: Proc. of the 10th International Planning Competition: Planner and Domain Abstracts – Hierarchical Task Network (HTN) Planning Track (IPC 2020). pp. 5–8 (2021)
9. Muise, C.: Planning.Domains. In: ICAPS – Demo 2016 (2016)
10. Schreiber, D.: Lilotane: A lifted SAT-based approachto hierarchical planning. Journal of Artificial Intelligence Research (JAIR) (70), 1117–1181 (2021)
11. Sleath, K., Bercher, P.: Experimental results for the PRICAI 2023 paper "Detecting AI planning modelling mistakes – potential errors and benchmark domains" (2023). https://doi.org/10.5281/zenodo.8249690