Rescue Robot World

Marcos Silveira
Pontifícia Universidade Católica do Rio Grande do Sul - PUCRS
bymarkone@gmail.com

Abstract

Many times researchers that starting to go a little bit deeper on their studies on classical planners want to develop planners, either for having a general understanding of how a planner work or planning algorithm work, either for implementing specific planning feature like the many we observed being incorporated on planning competitions each year. What usually happens in such attempts is that the researcher will spend most of the energy writing the parser for the planning definition language (usually PDDL) that will support her experiments, and few or no time left to the experiment itself. This paper presents the results of building a parser for the PDDL 3.1 complete language, using Java, and building an object-oriented planning model that represents the planning domain and problem. We believe that the use of such a tool will allow students to focus on their experiment specifics, without putting too much effort on the parsing.

Introduction

Planning definition languages, more specifically PDDL, evolved constantly since its first appearance. Almost every year (or every competition) a new set of features to the language was presented with the goal of improving the range of problems that planners would be able to solve.

PDDL first version, 1.2, included the basic constructs of planning representation: *requirements*, *types*, *constants*, *predicates*, *actions* with its *parameters*, *preconditions* and *effects*, are the most important. [Ghallab, 1998]

PDDL 2.1 extended the language to include *numeric expressions*, *metrics* that allow a quality of a plan to be evaluated, and *durative actions* – a way of modeling temporal properties of a domain [Fox and Long, 2003]. Version 2.2 added, on the top of 2.1, the features of *derived predicates* and *timed initial literals* [Edelkamp and Hoffman, 2004].

In 2006, PDDL 3.0 was introduced to allow the expression of *strong and soft constraints on plan trajectories*, as well as *string and soft problem goals* [Afonso and Long, 2006]. And version 3.1, released in 2008 and still the latest version of PDDL, added to language the feature of *object-fluents* and *action costs* [Helmert, 2008].

Researchers have many times the need of building experiments on the top of these planning features. Their goals are usually varied: building planners that will take advantage of an specific feature, or testing new algorithms for different kinds of problems, or simply get to a deeper understanding of classical planning by having a change of building and debugging well established models of classical planning problem solvers.

While in purse of these goals, researches come across the problem that no reusable parsers are offered whenever a new version of PDDL is released. Despite that fact that many published planners have a parsing mechanism within, this mechanism is not detached in a way that it can be used by itself. In this context, it becomes imperative to build a parser before starting to work on planners and algorithms, and that is a time consuming task.

In this paper we present a parser, in Java, for the PDDL 3.1 language that will encapsulate the parsing phase, and offers a coherent planning object oriented model. ???

We are using the PDDL 3.1 BNF as published in ???

The paper will present the resulting model.

Experimentation

In order to validate both the domain and the examples exposed in the previous sections, we've executed the problems with two planners.

FastDownward is a planner know by it's focus on performance. It can use pluggable search heuristics that will be suitable to each of the problem domain being dealt with.

Java GP is a planner built with didactical purposes. It's main focus is on teaching and not performance.

Table 1 – Total Execution time of the problems

	Fast Downward	Java GP
First Problem	0 s	0.372 s
Second Problem	0 s	1.349 s
Third Problem	0 s	4h + Not finished

Table 2 - Memory use of the problem

	Fast Downward	Java GP
First Problem	611796 KB	131500 KB
Second Problem	610772 KB	166000 KB
Third Problem	611796 KB	3680000 KB

Table 1 shows the time execution of both fast downward and Java GP. As it can be noticed, FastDownward is much faster. The planner logs show that all three problems were solved in time smaller than 1 second.

Java GP is much slower. The third execution didn't finish after running for 4 hours.

Table 2 shows memory use for both planners.

References

[Howe, 2006] J. Howe, "Crowdsourcing: A definition," http://crowdsourcing.typepad.com/cs/2006/06/crowdsourcing a.html, June 2006.

[Russel and Norvig, 2003] Stuart J. Russell, Peter Norvig. Artificial Intelligence: A Modern Approach. 2 ed., Upper Saddle River, New Jersey: Prentice Hall, 2003, pp. 375– 459

[Ghallab, 1998] Ghallab, Malik, et al. "PDDL-the planning domain definition language." (1998).

[Fox and Long, 2003] Fox, Maria, and Derek Long. "PDDL2. 1: An Extension to PDDL for Expressing Temporal Planning Domains." J. Artif. Intell. Res.(JAIR) 20 (2003): 61-124.

[Edelkamp and Hoffman, 2004] Edelkamp, Stefan, and Jörg Hoffmann. "PDDL2. 2: The language for the classical part of the 4th international planning competition." 4th International Planning Competition (IPC'04), at ICAPS'04 (2004).

[Afonso and Long, 2006] Gerevini, Alfonso, and Derek Long. "Preferences and soft constraints in PDDL3." ICAPS workshop on planning with preferences and soft constraints. 2006.

[Helmert, 2008] Helmert, M. "Changes in PDDL 3.1." Unpublished summary from the IPC-2008 website (2008).

[Helmert, 2006] Helmert, Malte. "The Fast Downward Planning System." J. Artif. Intell. Res.(JAIR) 26 (2006): 191-246.

[Mao, 2015] Mao, Ke, et al. "A Survey of the Use of Crowdsourcing in Software Engineering." RN 15 (2015): 01.