

Knowledge Engineering Tools for Planning in PDDL - Syntax Highlighting, Task Generation and Plan Visualization and Execution - an extensible framework

Volker Strobel

March 4, 2014

Contents

1	Introduction	3
2	Related Work	4
2.1	PDDL Studio	4
2.2	PDDL Mode for Emacs	5
2.3	itSIMPLE	5
2.4	GIPO	5
2.5	ModPlan	5
2.6	VISPLAN	5
2.7	Conclusion	5
3	Planning Basics and PDDL	6
3.1	Domain File	7
3.1.1	Define	7
3.1.2	Requirements	7
3.1.3	Types	8
3.1.4	Functions	8
3.1.5	Actions	8
3.2	Problem File	8
3.3	Planning	8

4	Software Engineering Tools for AI Planning	9
4.1	Statement of Problem	9
4.2	Syntax Highlighting and Code Snippets	10
4.2.1	Implementation	10
4.2.2	Usage and Customization	11
4.2.3	Evaluation	12
4.3	Clojure Interface	12
4.3.1	Basics	12
4.3.2	Functions	12
4.3.3	Numerical Expressiveness	12
4.4	Type Diagram Generator	15
5	Analysis	16
5.1	Participants	16
5.2	Material	16
5.3	Design	16
5.4	Procedure	16
6	Conclusion and Outlook	16
6.1	Outlook	17
7	Appendix	18

Abstract

Automated planning and scheduling blazes a trail for artificial intelligent behavior. PDDL, a planning language widely used for AI task specifications requires extensive knowledge engineering effort. After giving an overview of PDDL syntax and planning basics, a plug-in (including syntax highlighting) for the code editor Sublime Text and related editors will be given. A type diagram generator will be presented that supports modeling of the environment. The usability of these tools will be evaluated by subjects without prior knowledge in PDDL. By the use of these tool, the average modeling time of a shorter task specification could be improved by ... percent and the error rate could be reduced from ... to Furthermore, this thesis will present approaches for the automatic generation of task specifications and provide a basic interface between the functional programming language Clojure and PDDL. The proposed tools can assist knowledge engineers in the design process.

<https://www.ece.cmu.edu/~koopman/essays/abstract.html>

1 Introduction

TODO:

- Scope of the article (What did I miss out, what is included?)
- Name Tools and provide found possibilities and limitations
- Motivation: Why is this article interesting?
- Structure of the article
- Review of relevant literature? Note: I can review the relevant literature and related work in the related chapter, if this is convenient
- Planning Architecture - 3 Tier Architecture
- ICAPS
- Handicapped people -> Planning Interaction
- Why is something interesting? -> reasons! (and not because it does not exist yet)

Planning is one the the classic Artificial Intelligence (AI) tasks. Knowledge engineering (KE), that means representing world information in a computer system, is the crucial step for utilizing AI planning systems to solve problems. By definition (TODO: ...), it requires an human expert who knows the underlying syntax and models the information manually. This process is naturally error-prone and time-consuming. While planning systems are improving steadily (...), the main work for useful systems lies on the representing language. The Planning Domain Definition Language (PDDL) (McDermott et al. 1998) allows for a standardized way of specifying planning tasks. While on the one hand, recent PDDL extensions (Fox

and Long 2003; Kovacs 2011) extended the expressiveness of PDDL and tackle a route for real-world applications, they also demand a higher level of knowledge and attention of the knowledge engineer. Particularly with the start in 2005 of the International Competition on Knowledge Engineering for Planning and Scheduling (ICKEPS), new tools for the shift from at the modeling process from a text based 'programming paradigm' to a user-friendly graphical design tool exist, however, they also arouse drawbacks: limited functionality, expenditure of time, editing difficulty, to name a few. As in other computer languages, so far, there is no real way around diving into the 'code paradigm'. Despite recent advances in the area, the performance of planning & scheduling systems is still dependent to a large extent on how problems and domains are formulated, resulting in the need for careful system fine-tuning (copied 1:1). This thesis will, after introducing the basic terms and definitions of PDDL, focus on the software support for knowledge engineers. A package for the text and source code editor Sublime Text (ST) will be introduced, that provides syntax highlighting and PDDL templates. And the functionality is transferred to a on-line editor with instant access. Together with this package, a basic interface between the functional programming language Clojure and PDDL is developed. In this process, a UML 'type diagram' generator, derived from UML class diagrams for Object Oriented Programming, is presented that supports the knowledge engineer in keeping in keeping track during the design process and gain a fast overview of existing domains and problems.

The main focus of this thesis is about real world applications and the development of handy tools that support (and partially automatize) the planning process.

2 Related Work

2.1 PDDL Studio

PDDL Studio (Plch et al. 2012) is an Integrated Development Environment (IDE) for creating PDDL tasks (domains and problems) with supporting editing features that are based on a PDDL parser, like syntax highlighting, code collapsing and code completion. It provides a sophisticated on the fly error detection, that splits errors in syntax errors, semantic errors and errors found during parsing. The code completion feature allows completion for standard PDDL constructs and dynamic list completions, that were used in the current project (TODO: technical terms!).

Tasks are organized into project, that means a project is composed of a domain and associated problem(s).

An interface allows the integration of command line planners in order to run and compare different planning software.

Furthermore it provides a XML Export/Import feature (i.e. XML->PDDL, PDDL->XML) and further common editor features like a line counter, bracket matcher and a auto-save feature.

that means syntax and semantic checking, syntax highlighting, code completion and project management. While colors for highlighted code can be customized, the background color of the tool is always white.

2.2 PDDL Mode for Emacs

PDDL-mode is a major Emacs mode for browsing and editing PDDL 2.2 files. It supports syntax highlighting by basic pattern matching, regardless of the current semantic, automatic indentation and completions. Code snippets for the insertion of domains, problems and actions are provided. A declaration menu shows all actions and problems in the current PDDL file.

2.3 itSIMPLE

The itSIMPLE project is a tool that supports the knowledge engineer in designing a PDDL domain by the use of UML diagrams. This approach is reversed to the approach mentioned in this paper: while itSIMPLE generates PDDL from UML, this paper generated UML from PDDL.

2.4 GIPO

2.5 ModPlan

Also see VEGA plan visulazation on the MODplan page

- Very interesting: <http://www.tzi.de/~edelkamp/modplan/>

2.6 VISPLAN

2.7 Conclusion

As it can be seen, there is need for an up-to-date, customizable, text editor with PDDL support, that supports the current standard PDDL 3.1.

3 Planning Basics and PDDL

- Brief summary at start
- Start with a paragraph that describes the context
- Very interesting for basics of PDDL:
- <http://www.ida.liu.se/~TDDC17/info/labs/planning/writing.html>
- Konstruktionsanleitung
- Propositionale Logic -> Artificial Intelligence a Modern Approach
- To insert somewhere:
 - It should be mentioned, that almost no planner supports every part of PDDL. And, additionally, the quality of error messages is very diversified. While some simple state: error occurred, other list the problem and the line.

Introduction to planing: http://books.google.de/books?id=eCj3cKC_3ikC&printsec=frontcover&dq=automated+planning&hl=en&sa=X&ei=3wgNU5fQIcHx4gSTsoDABA&res=y#v=onepage&q=automated%20planning&f=false

AI planning describes ...

A planner and use the generated solution file (*plan*).

PDDL was first described in PDDL-the planning domain definition language (1998) and has been in constant development since then. This thesis makes use of *PDDL 3.1* (n.d.) if not otherwise stated.

PDDL planning task specifications are composed of two separate text files:

- Domain file: description of general types, predicates, functions and actions -> uninstantiated problem independent

- Problem file: description of a concrete problem environment -> instance specific

This separation allows for an intuitive process of task modeling: While general instances are described in the domain file, specific instances of problems are created in the problem files.

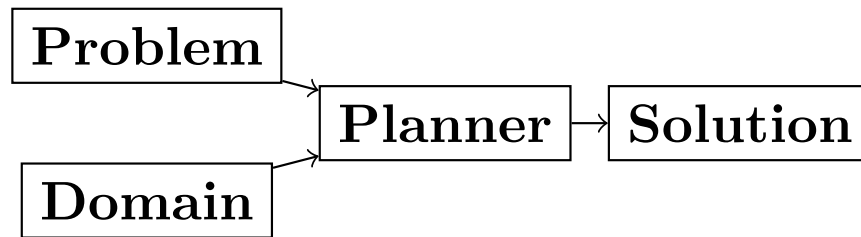


Figure 1: PDDL Planning workflow

These two files shall be investigated further in the following sections.

3.1 Domain File

The domain file contains the frame for planning tasks and determines, which types, predicates and actions are possible

Domain files have a strict format: All keyword arguments must appear in the order specified in the manual (an argument may be omitted) and just one PDDL definition (of a domain, problem, etc.) may appear per file. Fox and Long 2003, p. 6.

Include simple domain -> \LaTeX Include simple problem -> \LaTeX Include simple plan -> not yet in \LaTeX

3.1.1 Define

Every domain file starts with (define (domain <domainName>) ...) where, <domainName> can be any string.

3.1.2 Requirements

The requirements part is not a mandatory part of a PDDL domain file. However, PDDL supports different "levels of expressivity", that means subsets

of PDDL features McDermott et al. (1998, p. 1). As most planners only support a subset of PDDL the requirements part is useful for determining if a planner is able to act on a given problem. They are declared by the `(:requirements ...)` part. Some often used requirements include `:strips`. For a list of current requirement flags and their meaning, see ...

3.1.3 Types

If order to be able to use types in a domain file, the requirement `:typing` should be declared (TODO: is `:adl` enough?).

In order to assign categories to objects, PDDL allows for type definitions. Like that, parameters in actions can be typed, as well as arguments in predicates, functions [extra source!]. Later, in the problem file, objects will be assigned to types, like objects to classes in Object Orientated Programming (OOP). Adding to the `(:requirement ...)` part of the file guarantees, that typing can be correctly used. Strips (no types) vs ADL (types).

3.1.4 Functions

Functions are not supported by many planners (source!) and, before PDDL 3.1 they could only be modeled as

It is notable that before PDDL 3.0 the keyword `functors` was used instead

3.1.5 Actions

PDDL 3.1 supports two types of actions: `durative-action` and the 'regular' action.

3.2 Problem File

Problems are designed with respect to a domain. Domains usually have multiple problems `p01.pddl`, `p02.pddl`, ... Problems declare the initial world state and the goal state to be reached. They instantiate types, in they way that they create objects

3.3 Planning

A planning solution is a sequence of actions that lead from the initial state to the goal state. PDDL itself does not declare any uniform plan layout.

The input to the planning software is a domain and a belonging problem, the output is usually a totally or partially ordered plan. are software tools

that Due to the yearly ICAPS, there is a broad range of available planners. This thesis uses the planner SGPLAN₆ Hsu and Wah (2008), a 'extensive' (in the sense of its supporting features) planner for both temporal and non-temporal planning problems.

An overview of different planners is given at <http://ipc.informatik.uni-freiburg.de/Planners>.

4 Software Engineering Tools for AI Planning

- PDDL type hierarchy and object instantiation to UML / TikZ, store predicates (and action?) in same box as type
- Research Knowledge Engineering in Planning
- Human Computer Interaction
 - <http://hci.waznelle.com/checklist.php>
- Write Tiago (itSimple) regarding PDDL -> UML (and knowledge engineering in general)
- ICKEPS (International Competition on Knowledge Engineering for Planning and Scheduling)
- Orient on "How to Design Classes"

4.1 Statement of Problem

Writing and maintaining PDDL files can be time-consuming and cumbersome Li et al. (2012). So, the following development tools shell support and facilitate the PDDL task design process and reduce potential errors.

Below, methods are presented for

Syntax Highlighting and Code Snippets Environment for Editing PDDL files

Class Diagram Generator The automation of the PDDL task design process. File input and output and dynamic generation (design level)

Human Planner Interaction An interactive PDDL environment: speech synthesis and recognition.

Domain Generator Mathematical limitations (design level)

4.2 Syntax Highlighting and Code Snippets

Writing extensive domain and problem files is a cumbersome task: longer files can get quickly confusing. Therefore, it is convenient to have a tool that supports editing these files. Syntax highlighting describes the feature of text editors of displaying code in different colors and fonts according to the category of terms (source: Wiki). A syntax highlighting plug-in for the text and source code editors Skinner (2013a) and Skinner (2013b) is proposed and transferred to the on-line text editor Ace are used to implement this feature, as ST Syntax Highlighting files can easily be converted to Ace Files.

For Mac user, TextMate (TM) is very similar to ST and the syntax highlighting file can be used there, too. Besides, the general principles (e.g. regular expressions) outlined here, apply to most of other editors as well. So, a Pygments extension was written, that allows for syntax highlighting in \LaTeX documents.

4.2.1 Implementation

ST syntax definitions are written in property lists in the XML format.

The syntax definition is implemented by the use of the ST plug-in *AAAPackageDev* (2014). So, the definitions can be written in YAML in converted to Plist XML later on. *AAAPackageDEVAAAPackageDev* (2014) is a ST plugin, that helps to create, amongst others, ST packages, syntax definitions and 'snippets' (re-usable code).

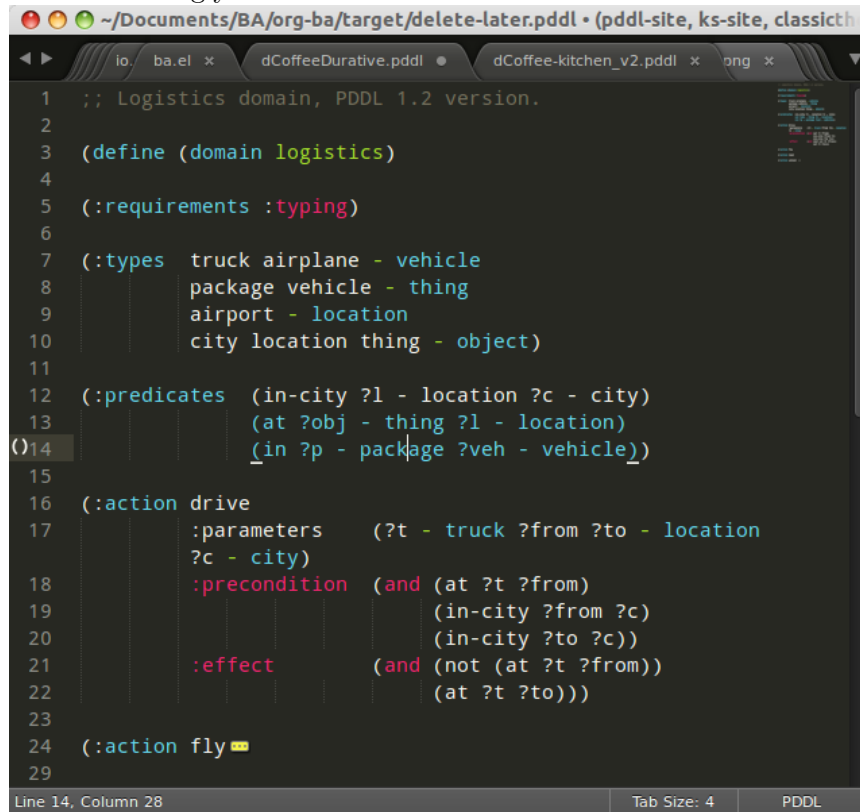
By means of Oniguruma regular expressions (Kosako 2007), scopes are defined, that determine the meaning of the PDDL code block. The scope naming conventions mentioned in the *TextMate 2 Manual* are applied here. By the means of the name, the colors are assigned. Different ST themes display different colors (not all themes support all naming conventions).

The syntax highlighting is intended for PDDL 3.1, but is downward compatible, as previous versions are subsets of later versions.

Are later versions really subsets?

nil

The pattern matching heuristic that is implemented by the use of regular expressions is used for assigning scopes to the parts of the file. This heuristic is quite sophisticated, as it recognizes sub-parts of PDDL files and chooses the color accordingly.

A screenshot of a Sublime Text editor window. The title bar shows the file path: ~/Documents/BA/org-ba/target/delete-later.pddl. The editor has several tabs open: 'io', 'ba.el', 'dCoffeeDurative.pddl', 'dCoffee-kitchen_v2.pddl', and 'ong'. The main editor area displays PDDL code with syntax highlighting. The code includes comments, domain definitions, requirements, types, predicates, and actions. Line 14 is highlighted. The status bar at the bottom shows 'Line 14, Column 28', 'Tab Size: 4', and 'PDDL'.

4.2.2 Usage and Customization

To enable syntax highlighting and code snippets in ST, the files of the repository have to be placed in the ST packages folder (<http://www.sublimetext.com/docs/3/packages.html>). The first part of the PDDL.YAML-tmlanguage describes the parts of the PDDL task that should be highlighted. By removing (or commenting) include statements, the syntax highlighter is adjustable to the user's need.

By using ST as editor, language independent ST features are supported,

like auto completion, code folding and column selection, described in the Sublime Text 2 Documentation.

The PDDL.YAML-tmlanguage file is split in two parts:

By default, all scopes are included.

4.2.3 Evaluation

4.3 Clojure Interface

PDDL, as planning language modeling capabilities are limited, a interface with a programming is handy a can reduce dramatically the modeling time. In IPC, task generators are used write extensive domain and problem files.

As PDDL's syntax is inspired by LISP (Fox and Long 2003, p. 64), using a LISP dialect for the interface seems reasonable. This thesis uses Clojure (Hickey 2008), a modern LISP dialect that runs on the Java Virtual Machine.

In this section, I will not only show a method for generating PDDL constructs, but also for reading in PDDL files are handling the input.

4.3.1 Basics

Through the higher-order filter method in Clojure, parts of PDDL files can be easily extracted. Like that, one can extract parts of the file and handle the constructs in a Clojure intern way.

As an example, the type handling will be represented here, but the basic approach is similar for all PDDL constructs.

The here developed tools should be platform independent with a development focus in UNIX/Linux systems, as most planners (source!) run on Linux.

4.3.2 Functions

As functions have a return value, the modeling possibilities dramatically increase.

4.3.3 Numerical Expressiveness

One might assume that the distance could be modeled as follows:

```
(durative action ...  
...  
  :duration (= ?duration (sqrt (coord-x )))  
...)
```

However, PDDL does only support basic arithmetic operations (+, -, /, *).

An Euclidean distance function that uses the square root would be convenient for distance modeling and measurement. However, PDDL 3.1 supports only four arithmetic operators (+, -, /, *). These operators can be used in preconditions, effects (normal/continuous/conditional) and durations. Parkinson and Longstaff (2012) describe a workaround for this drawback. By declaring an action ‘calculate-sqrt’, they bypass the lack of this function and rather write their own action that makes use of the Babylonian root method.

1. Alternative #1: Only sqrt exists Assuming that a function sqrt would actually exist, the duration could be modeled as follows:

```
:duration (= ?duration
            (sqrt
             (+
              (*
               (- (pos-x (current-pos))
                  (pos-x ?goal))
               (- (pos-x (current-pos))
                  (pos-x ?goal)))
              (*
               (- (pos-y (current-pos))
                  (pos-y ?goal))
               (- (pos-y (current-pos))
                  (pos-y ?goal)))))))
```

2. Alternative #2: sqrt and expt exist Assuming that a function sqrt would actually exist, the duration could be modeled as follows:

```
:duration (= ?duration
            (sqrt
             (+
              (expt
               (-
                (pos-x (current-pos))
                (pos-x ?goal)))
              (expt
               (-
```

```
(pos-y (current-pos))
(pos-y ?goal))))))
```

3. Alternative #3: Calculate distance and hard code it, e.g. (distance table kitchen) = 5.9

- Distance Matrix
- <http://stackoverflow.com/questions/20654918/python-how-to-speed-up-calculation>
- Scipy.spatial.distance (-> Clojure?)
- Mention that the Taxicab geometry allows different ways that have an equal length

Another alternative is to make use of an external helper and, instead of calculating every entry of the distance matrix. the distance only if needed, incorporate every possible combination of two locations. This approach has certainly a major drawback: With an increasing amount of locations, the number of combinations increases exponentially. That means, if there are 100 locations, there will be

TODO: Calculate possibilities

nil

... . The native approach would be to iterate over the cities twice and calculate only the half of the matrix (as it is symmetric, that mean distance from A to B is the same as the distance from B to A).

4. Alternative #4: Use the Manhattan distance

Allowing the agent to move only vertically and horizontally would be that one can use the so called Taxicab geometry (or Manhattan length) as distance measurement. In the Kitchen domain, this could be modeled as follows:

```
% => Metric: reduce duration
```

```
% dKitchenware.pddl
```

```
\begin{figure}[t]
```

```
\inputminted[mathescape, linenos, numbersep=5pt, frame=lines, framesep=2mm]
{csharp}
```

```

{Code/dKitchenware.pddl}
\caption{The basic kitchenware domain}
\end{figure}
\section

```

TODO:

TODO Human Planner Interaction

4.4 Type Diagram Generator

Add actions to the Type Diagram?

Graphical notations, have some advantages compared to textual notations, as they simplify the communication between developers and help to quickly grasp the connection of related system units.

But for all that one disadvantage has to be accepted:

The UML was invented in order to standardize modeling in software engineering (SE). The UML consists of several part notations, the here presented tool uses the 'class diagram' notation, as PDDL types and classes in OOP have strong resemblance (see Tiago 2006, p 535).

Types play a major role in the PDDL design process: they are involved, besides their definition, in the constants, predicates and actions part. So, a fine grasp of their hierarchy, as well as their involved predicates becomes handy and assists the KE in the planning process. Types strongly resemble classes in object oriented programming As mentioned in chapter (...), the type definitions follow a specific syntax. For example `truck car - vehicle` would indicate, that both `truck` and `car` are subtypes of the super-type `vehicle`.

Subtypes and corresponding super-types can be extracted using regular expressions. `#"(...)"` matches every kind of that form and a Clojure-friendly representation in form of a hash-map can be created.

PDDL side ————— Clojure side
`(':types ... — ...) {... [...], ...}`

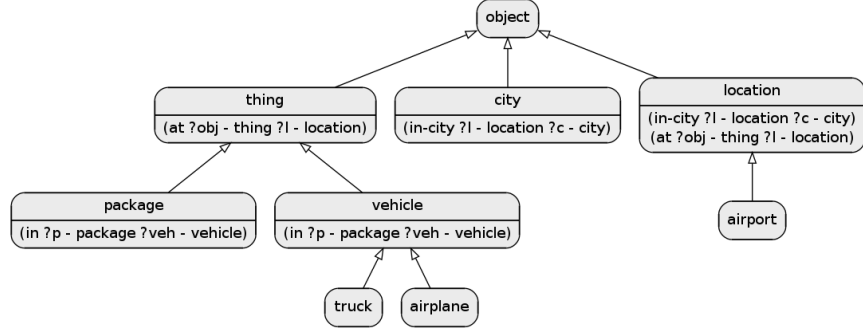


Figure 2: Part of a PDDL domain and the corresponding, generated UML diagram

5 Analysis

5.1 Participants

Ten non-paid students (six female) took part in the experiment. All had knowledge about LISP syntax, but neither one had faced PDDL prior to this study.

5.2 Material

The usability of the Syntax Highlighter (see 4.2) and the Type Diagram Generator (see Type Diagram Generator) were tested.

5.3 Design

The participants had to

5.4 Procedure

6 Conclusion and Outlook

The tools presented in this thesis have been designed to support knowledge engineers in planning tasks. They can support engineers in the early planning design process, as well as in the maintenance of existing domains and problems. The communication between engineers can be facilitated and

6.1 Outlook

Besides ICKEPS, as mentioned in the introduction, also the yearly workshop Knowledge Engineering for Planning and Scheduling (KEPS) will promote the research in planning and scheduling technology. Potentially, the main effort of for implementing models in planning will be shifted from the manual KE to the automated knowledge acquisition (KA). Perception systems, Nevertheless, a engineer who double-checks the generated tasks will be irreplaceable.

References

- [1] *AAAPackageDev*. 2014. URL: <https://github.com/SublimeText/AAAPackageDev/> (visited on 02/24/2014).
- [2] Maria Fox and Derek Long. “PDDL2. 1: An Extension to PDDL for Expressing Temporal Planning Domains.” In: *J. Artif. Intell. Res.(JAIR)* 20 (2003), pp. 61–124.
- [3] Rich Hickey. “The clojure programming language”. In: *Proceedings of the 2008 symposium on Dynamic languages*. ACM. 2008.
- [4] Chih-Wei Hsu and Benjamin W Wah. “The sgplan planning system in ipc-6”. In: *Proceedings of IPC*. 2008.
- [5] K. Kosako. *Oniguruma Regular Expressions Version 5.9.1*. 2007. URL: <http://www.geocities.jp/kosako3/oniguruma/doc/RE.txt> (visited on 02/24/2014).
- [6] DL Kovacs. “BNF definition of PDDL 3.1”. In: *Unpublished manuscript from the IPC-2011 website* (2011).
- [7] Yi Li et al. “Translating pddl into csp#-the pat approach”. In: *Engineering of Complex Computer Systems (ICECCS), 2012 17th International Conference on*. IEEE. 2012, pp. 240–249.
- [8] MacroMates Ltd. *TextMate 2 Manual*. 2013. URL: <http://manual.macromates.com/en/> (visited on 02/24/2014).
- [9] Drew McDermott et al. “PDDL-the planning domain definition language”. In: (1998).

- [10] Simon Parkinson and Andrew P Longstaff. “Increasing the Numeric Expressiveness of the Planning Domain Definition Language”. In: *Proceedings of The 30th Workshop of the UK Planning and Scheduling Special Interest Group (PlanSIG2012)*. UK Planning and Scheduling Special Interest Group. 2012.
- [11] *PDDL 3.1*. URL: <http://ipc.informatik.uni-freiburg.de/PddlExtension>.
- [12] Tomas Plch et al. “Inspect, edit and debug pddl documents: Simply and efficiently with pddl studio”. In: *and Exhibits* (2012), p. 15.
- [13] Jon Skinner. *Sublime Text 2*. <http://www.sublimetext.com/2>. Release 2.0.2. 2013.
- [14] Jon Skinner. *Sublime Text 3*. <http://www.sublimetext.com/3>. Beta Build 3059. 2013.

7 Appendix

α