Automatic Planning: From Event-B to PDDL

Sabrine Ammar(✉) and Mohamed Tahar Bhiri

Faculty of Sciences Sfax, Sfax University, Sfax, Tunisia
ammar.sabrine@hotmail.fr, tahar_bhiri@yahoo.fr

Abstract. Automatic planning is a separate discipline of Artificial Intelligence (AI). It aims to formalize the planning problems described by the concept of state space. The Planning Domain Definition Language (PDDL) is a de facto standard language in the field of automatic planning. PDDL-related dynamic analysis tools, namely planners and validators, are insufficient for verifying and validating PDDL descriptions. Such tools make it possible to detect errors a posteriori by means of a test activity. In this article, we recommend a rigorous approach coupling Event-B and PDDL for automatic planning. Event-B is used for formal modeling by stepwise refinement with mathematical proofs of planning problems. A refinement strategy appropriate to planning problems is, then, proposed. The ultimate Event-B model, correct by construction, supposed to be translatable into PDDL, is automatically translated into PDDL using our MDE Event-B2PDDL tool. The obtained PDDL description is submitted to efficient planners for generation of solution plans.

Keywords: Automatic planning · PDDL · Event-B · Correct by construction
Planner · State space · State change operator · MDE

1 Introduction

Automatic planning can describe and solve planning problems. It is applied in various fields such as robotics, management projects, Internet browsing, managing crisis situations, logistics and games. In an informal way, a planning problem can be described by a state space. A state models a stable situation of the processed planning problem. It can be an initial state, final state (also called goal) or intermediate state. Moving from one state to another is governed by transitions. Each transition is labeled by an action. It has a specification showing two parts: its condition of applicability and its effect. A planning problem can accept zero or many solutions. A solution called plan-solution is a sequence of actions that leads from the initial state to the goal state.

The automatic planning community has developed a formal de facto standard Planning Domain Definition Language (PDDL) [1, 2] to formally describe planning problems. In addition, this community has developed solvers (so-called planners) able to calculate solutions to PDDL-formalized planning problems. In addition, it has developed validation tools for verifying whether a given plan-solution can be derived from a PDDL description. In general, PDDL descriptions are difficult to write, read, and evolve. Moreover, the tools associated with the PDDL language, namely the planners and validators, do not allow a rigorous **a priori** analysis of the PDDL

E. H. Abdelwahed et al. (Eds.): MEDI 2018 Workshops, CCIS 929, pp. 247–254, 2018.
https://doi.org/10.1007/978-3-030-02852-7_21

descriptions. Indeed, these tools are used **a posteriori** after establishing PDDL descriptions.

In this work, we advocate the opening of the automatic planning community on the formal methods community through Event-B [3]. To achieve this, we suggest a transformation from Event-B to PDDL. This promotes the development correct by construction [3] of planning problems. The ultimate Event-B model, derived from a chain of refinements with mathematical proofs, is translated into PDDL in order to generate **quality** plans through various planners supporting PDDL.

This article has five sections and one conclusion. The second section presents and evaluates the PDDL language and verification and validation tools associated with this language. The third section proposes an Event-B and PDDL coupling approach. The fourth section provides a refinement strategy for formal modeling of Event-B planning problems. The ultimate Event-B model stemming from this strategy is supposed to be translatable into PDDL. Finally, the fifth section describes our MDE Event-B2PDDL tool. The conclusion draws up the balance sheet of this article and proposes the possible extensions of this work.

2 Planning in Artificial Intelligence

Planning is a separate discipline of AI: planning community with dedicated conferences such as ICAPS (International Conference on Planning and Scheduling). It aims to **formalize** the planning problems described by the concept of state space. Thus, formal languages based on the logic of first-order predicates are proposed and enriched within the framework of the IPC-International Planning Competitions. In the following, we present the fundamental aspects of PDDL considered as de facto standard language in the field of AI planning.

2.1 The PDDL Language

A planning problem formalized using PDDL has two separate parts: domain and problem. The domain construction offered by PDDL makes it possible to describe all the aspects common to a class of problems known as generic domain. The AI planning community, in the IPC framework, has identified more than 50 domains grouped into four broad categories: logistics, robotics, gaming, and business applications. A domain described in PDDL includes types, constants, predicates, numeric functions and actions.

As an example, Listing 1 from [4] describes in PDDL the domain of the sliding puzzle game. The domain of the sliding puzzle game has two types: position and tile. In PDDL, a type does not have a structure and is designated by a name. The predicates at (having two parameters ?tile type tile and ?position type position), neighbor and empty allowing to formalize the concept of state of a sliding puzzle game problem.

Listing 1. State of the application

```
(define (domain n-sliding-puzzle)
 (:types position tile)
 (:predicates (at ?tile -tile ?position -position)
              (neighbor ?p1 -position ?p2 -position)
              (empty ?position -position))
 (:action move
    :parameters (?tile -tile ?from ?to -position)
    :precondition (and (neighbor ?from ?to)
                  (at ?tile ?from) (empty ?to))
    :effect (and (at ?tile ?to) (empty ?from)
                 (not (at ?tile ?from)) (not(empty ?to)))
 )
```

Listing 2. Sliding puzzle game with 8 tiles

```
(define (problem n-sliding-puzzle-bootstrap-33-01)
 (:domain n-sliding-puzzle)
 (:objects p_1_1 p_1_2 p_1_3 p_2_1 p_2_2 p_2_3 p_3_1
           p_3_2 p_3_3 -position t_1 t_2 t_3 t_4 t_5 t_6
           t_7 t_8 -tile)
 (:init
   ;; initial position of the tiles
  (at t_4 p_1_1) (empty p_1_2) (at t_8 p_1_3)
  (at t_6 p_2_1) (at t_3 p_2_2) (at t_2 p_2_3)
  (at t_1 p_3_1) (at t_5 p_3_2) (at t_7 p_3_3)
   ;;framework definition
  (neighbor p_1_1 p_1_2) (neighbor p_1_2 p_1_1)
  (neighbor p_1_2 p_1_3) (neighbor p_1_3 p_1_2)
  (neighbor p_2_1 p_2_2) (neighbor p_2_2 p_2_1)
  (neighbor p_2_2 p_2_3) (neighbor p_2_3 p_2_2)
  (neighbor p_3_1 p_3_2) (neighbor p_3_2 p_3_1)
  (neighbor p_3_2 p_3_3) (neighbor p_3_3 p_3_2)
  (neighbor p_1_1 p_2_1) (neighbor p_2_1 p_1_1)
  (neighbor p_1_2 p_2_2) (neighbor p_2_2 p_1_2)
  (neighbor p_1_3 p_2_3) (neighbor p_2_3 p_1_3)
  (neighbor p_2_1 p_3_1) (neighbor p_3_1 p_2_1)
  (neighbor p_2_2 p_3_2) (neighbor p_3_2 p_2_2)
  (neighbor p_2_3 p_3_3) (neighbor p_3_3 p_2_3))
 (:goal (and
   ;; final position of the tiles
  (at t_1 p_1_1) (at t_2 p_1_2) (at t_3 p_1_3)
  (at t_4 p_2_1) (at t_5 p_2_2) (at t_6 p_2_3)
  (at t_7 p_3_1) (at t_8 p_3_2))))
```

In PDDL, an action can have parameters typed (**parameters** clause) and defined by a Pre/Post specification: the two **precondition** and **effect** clauses. An operator (action in PDDL) can be applied in a state if and only if all pre-conditions are satisfied in this state. The effect of a PDDL action is defined by the additions and withdrawals of atoms in the current state.

The construction **problem** shown in PDDL makes it possible to formalize a problem belonging to the domain described by the construction **domain**. A problem formalized in PDDL includes the domain of this problem, typed objects (objects), an initial state (init) and a goal state (goal). For example, the Sliding Puzzle game containing 8 tiles from [4] is shown in Listing 2.

A planning problem described using PDDL is solved by a software item called planner. The AI planning community has developed several planners such as FF, LPG [5] and FD. A planner combines exploration and logic. Indeed, it can be seen either as a program that calculates a solution called plan-solution or as a program that **demonstrates** the existence of a solution. For example, the Sliding Puzzle game planning problem described by the two Listings 1 and 2 submitted to the LPG planner provides a plan-solution comprising 52 actions, an extract of which is shown in Listing 3.

Listing 3. Plan-solution extract associated with the 8-tile problem of the Sliding Puzzle game

```
0:      (MOVE T_8 P_1_3 P_1_2)
1:      (MOVE T_2 P_2_3 P_1_3)
2:      (MOVE T_3 P_2_2 P_2_3)
3:      (MOVE T_8 P_1_2 P_2_2)
4:      (MOVE T_2 P_1_3 P_1_2)
5:      (MOVE T_3 P_2_3 P_1_3)
6:      (MOVE T_8 P_2_2 P_2_3)
7:      (MOVE T_6 P_2_1 P_2_2)
8:      (MOVE T_4 P_1_1 P_2_1)
9:      (MOVE T_2 P_1_2 P_1_1)
```

2.2 Evaluation

PDDL offers interesting ways to represent planning problems. Indeed, PDDL supports various representations such as propositional representations, first order logic, both numeric and temporal. This makes it possible to describe the **states** and **actions** of a planning problem. The tools associated with the PDDL language are: planners and validators. Unlike a planner who performs a plan-solutions **production** activity, a validator [6] performs a **verification** activity. From the functional point of view, a validator accepts as input: a PDDL description (**domain** and **problem** file) and one or more plan-solutions files and outputs a verdict. A 'YES' means that plan-solutions can be obtained from the subject PDDL description. A 'NO' means a failure. Validators can be used with profit to appreciate PDDL domains by adopting the functional test. In addition, the validators allow verification by checking the plan-solutions generated by

various planners. Finally, a validator can be used as a tool to **objectively** compare the abilities of various planners. The **dynamic** analysis tools associated with PDDL, namely planners and validators, are insufficient for the verification and validation of PDDL descriptions. Indeed, complex PDDL descriptions involving actions with elaborated preconditions and postconditions are prone to errors that are hard to detect **a priori**. In fact, the dynamic analysis tools associated with PDDL makes it possible to detect errors **a posteriori** by means of a test activity.

3 From Event-B to PDDL

The formal Event-B method supports both horizontal and vertical refinement techniques. It allows the modeling of various domains: sequential programs, concurrent programs, distributed programs, reactive systems and recently hybrid systems. It has a platform called Rodin [7] based on Eclipse, including tools for verification (mathematical provers), validation (model-checker, animators and simulators) and code generation. Introduced by Jean-Raymond Abrial, the Event-B method is an evolution of B method [8]. Event-B is used to formally describe systems and reason mathematically about their properties. Event-B supports modeling, correction (or proof) and validation activities. These complementary activities characterize the development of Event-B systems. An Event-B model can only contain contexts (construction CONTEXT), only machines (construction MACHINE) or both. In the first case, the model represents a purely mathematical structure. In the third case, the model is parameterized by the contexts. Finally, the second case represents a model that is not parameterized. Contexts are modeling static properties of the model. Machines (construction MACHINE) are modeling the dynamic behavior of the system. A machine may refine (REFINES relation) and see another one or more contexts (SEES). The state of the machine is defined by variables introduced by the VARIABLES clause. The invariance properties related to these variables are grouped together in the INVARIANT clause. An Event-B machine groups events that affect its state. An event consists of two parts: a "guard" that defines the condition according to it the event may or may not be triggered, and an "action" called body for evolving state variables.

We advocate a rigorous approach combining Event-B and PDDL for automatic planning. Event-B is used for formal modeling by successive refinements with mathematical proofs of planning problems. The refinement of data supported by Event-B can be used profitably to refine the notion of state of a planning problem **step-by-step**. In addition, the one-to-many refinement shown in Event-B is very useful for **determining** the state change operators of a planning problem. Finally, the possibility of reinforcing the guard of an Event supported by Event-B during a refinement step is very useful for incrementally identifying the conditions of applicability of a state change operator of a planning problem.

Proof tools associated with Event-B (generator of proof obligations and provers) guarantee in particular the verification of the consistency of a planning problem described by Event-B. The ProB [9] tool that accepts Event-B offers the possibility of checking the dynamics of a planning problem by using the LTL^{e} language to specify temporal properties.

The use of Event-B coupled to ProB allows to obtain Event-B model correct by construction (thanks to the Event-B theory: proof obligations) and valid (thanks to ProB) describing a planning problem. Then we have to translate this Event-B model into a PDDL. To achieve this, several refinement steps are required in order to have a model described by **a subset of Event-B**: the data are described by the language of the first-order predicates of Event-B (the theory of sets is discarded because it is not translatable to PDDL) and the processing are described only through deterministic action (:=).

4 Proposed Refinement Strategy

Following numerous Event-B modeling of various planning problems, we have established a refinement strategy that could be reused to model in Event-B various planning problems in several areas [10]. Indeed, all planning problems can be formalized by the concept of **state space**: initial state, goal states, intermediate states and state change operators. Based on all of its common aspects of planning problems, we propose the refinement strategy that includes the steps outlined and justified below.

Step 1: Initial abstract model
The initial abstract model of a planning problem includes elements related to the notion of state, the initial state, and the goal states. These elements are formalized respectively in Event-B by typed variables and having invariant properties, INITIALISATION event and an event called goal having a guard to see if the current state is a goal state. The goal event does nothing (skip action). In addition, the initial abstract model of a planning problem must involve an overly abstract and non-deterministic modeling of the notion of state change operator. This is made possible by the ANTICIPATED status of an Event-B event.
Step 2: Determination of actions by successive refinements
This step includes several successive refinements allowing, ultimately, obtaining an Event-B model with state change operators having deterministic behaviors: The actions ($:\in$ and $:|$) are concretised. Each operator contains a guard modeling the condition of applicability of the operator and its action. Refinement techniques supported by Event-B as an event decomposition (one to many) and the strengthening guards are very useful for implementing this step. The state change operators are modeled by events in Event-B whose guards indicate the conditions of application of these operators and the actions that are modeling the changes of state: transition from one state to another in state spaces. To list all of the state change operators related to application, we recommend using parameterized non-deterministic events.
Step 3: Determination of parameters by successive refinements
This step aims to remove the non-determinism related to the parameters introduced in the clause ANY of each operator of states changes. Eventually, we obtain events without parameters. Technically, in this step, the one to many refinement technique and the WITH clause are used with advantage.
Step 4: Reinforcing the conditions of applicability

This step consists in reinforcing the conditions of applicability of the state change operators (WHERE clause) introduced in the previous step. The ultimate model from this step must have state change operators with rigorous semantics. Technically, this step introduces new invariant properties (reinforcement of the invariant) and guards (reinforcement of guards).

Step 5: Realization of data by successive refinements

The purpose of this step is to eventually provide an Event-B model translatable into PDDL. All Event-B set constructions must be realized using the Event-B predicative constructions. To achieve this, data refinement is used via Event-B gluing invariant.

Step 6: Conveying a **reduced** Event-B into PDDL

The reduced Event-B model from Step 5 is translated using our Event-B2PDDL tool introduced in Sect. 5.

This refinement strategy has been successfully applied to the problem of three cannibals and three missionaries.[1]

5 The Event-B2PDDL Tool

Our Event-B2PDDL tool takes as input a reduced Event-B model that is translatable into PDDL and outputs a PDDL description acceptable to planners. Event-B2PDDL is based on simple intuitive rules allowing the systematic translation of Event-B elements to PDDL elements. Event-B2PDDL is made according to MDE technology.

5.1 Event-B to PDDL Transformation Rules

The PDDL description from the Event-B2PDDL tool has two domain and problem constructions (see Sect. 2.1). Thus, in [10], we have respectively established rules allowing the translation of Event-B elements related to the planning domain and the planning problem.

The translation rules for Event-B elements related to the planning domain concern: the translation of abstract sets, constants, Boolean constants or variables, Boolean functions, events and formulas.

The rules for conveying Event-B elements related to the planning problem concern the translation of the constants linked to the sets defined by enumeration and the translation of two INITIALISATION and GOAL events.

5.2 Translation Automation from Event-B to PDDL

Using both MDE Xtext and Xtend tools, we developed the Event-B2PDDL tool based on the transformation rules presented in 5.1. The Xtext tool allowed us to design a DSL for our input language: A reduced Event-B with only those constructions that are taken into account by the transformation. Transformation and generation of PDDL code is programmed in Xtend.

[1] https://crocodeal.tn/startbootstrap-resume-gh-pages/.

6 Conclusion

In this work, we proposed an Event-B to PDDL coupling approach. The transition from Event-B to PDDL makes it possible to model correct by construction and efficient planning problems. Event-B ensures the correct by construction of the states change operators. Whereas PDDL ensures the effectiveness of the plan-solutions obtained thanks to the planners associated with PDDL. We proposed, in addition, a refinement strategy which may be appropriate for any planning problem that favors Event-B/PDDL coupling. The transformation of Event-B to PDDL gave rise to an MDE Event-B2PDDL tool. Currently, we are working in two directions: experimentation of the refinement strategy proposed in Sect. 4 on various more or less complex planning problems and development of refinement schemes allowing the realization of Event-B data in PDDL (from set representations to predictive representations). Eventually, such schemes could be automated by adopting the technique of automatic refinement like the BART tool [11] associated with the formal method B.

References

1. McDermott, D., et al.: PDDL-the planning domain definition language. Technical Report CVC TR- 98-003/DCS TR-1165, Yale Center for Computational Vision and Control, New Haven, CI, USA (1998)
2. International Conference on Automated Planning and Scheduling. www.icaps-conference.org
3. Abrial, J.-R.: Modeling in Event-B: Systems and Software Engineering. Cambridge University Press, New York (2010)
4. Bibai, J.: Segmentation et évolution pour la planification: le système Divide-And-Evolve. Université Paris Sud, Paris XI (2010)
5. Gerevini, A., Saetti, A., Serina, I.: User Instructions for LPG-td. http://burglar-game.googlecode.com/svn/branches/burglargameant/planner/lpg/README-LPGTD
6. Howey, R., Long, D., Fox, M.: VAL: automatic plan validation, continuous effects and mixed initiative planning using PDDL. In: Tools with Artificial Intelligence, ICTAI (2004)
7. Voisin, L., Abrial, J.R.: The rodin platform has turned ten. In: Ait Ameur, Y., Schewe, K.D. (eds.) ABZ 2014. LNCS, pp. 1–8. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-43652-3_1
8. Abrial, J.-R.: The B-Book: Assigning Programs to Meanings. Cambridge University Press, New York (1996)
9. Leuschel, M., Butler, M.: ProB: a model checker for B. In: Araki, K., Gnesi, S., Mandrioli, D. (eds.) FME 2003. LNCS, vol. 2805, pp. 855–874. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-45236-2_46
10. Fourati, F.: Contributions à l'analyse statique et dynamique d'architectures logicielles. Faculty of Science of Sfax, Tunisia (2017)
11. Requet, A., BART: a tool for automatic refinement. In: ABZ, p. 345 (2008)