Artificial Intelligence

15. PDDL

How to Explain Your Problems to a Computer

Prof Sara Bernardini
bernardini@diag.uniroma1.it
www.sara-bernardini.com

SAPIENZA
UNIVERSITÀ DI ROMA

Autumn Term

Agenda

PDDL

What is PDDL?

- Once you decided for STRIPS/FDR/whatever, you still need to design an input syntax that your computer can read.
- That input syntax in the planning area is PDDL: The Planning Domain Definition Language.
- In particular, PDDL is used in the International Planning Competitons (IPC).

Why PDDL? It's just a fact of life:

→ PDDL is the de-facto standard input language in the planning area.

→ To complete this course you must know this language.

(Initially, every group used their own input language = needing an interpreter every time you talk to your neighbor.)

Our Agenda for This Chapter

2. **Schematic Encodings:** Explains the main design principle behind PDDL.
3. **PDDL Grammar:** Outlines the syntax, with example snippets.
4. **History and Extensions:** Summary of what's out there and how we got there.

Schematic Encodings

Schematic encodings use variables that range over objects:

- Predicates instead of STRIPS propositions. Arity: number of vars.
- Action schemas instead of STRIPS actions. Arity: number of vars.

- Analogy: propositional logic vs. predicate logic (FOL).
- Set of objects in PDDL is finite!

$\rightarrow$ Like FOL, PDDL describes the world in a schematic way relative to a set of objects. This makes the encoding *much* smaller and easier to write.

$\rightarrow$ Most planners translate the schematic input into (propositional) STRIPS in a pre-process, by instantiating the variables in all possible ways. This is called grounding.

Schematic Actions: Example

The schematic action (triple means (pre, add, del)):

$$x \in \{car1, car2\}$$
$$y_1 \in \{London, Edinburgh\},$$
$$y_2 \in \{London, Edinburgh\}, y_1 \neq y_2$$
$$(\{at(x, y_1)\}, \{at(x, y_2)\}, \{at(x, y_1)\})$$

corresponds to the actions:

$$(\{at(car1, London)\}, \{at(car1, Edinburgh)\}, \{at(car1, London)\}),$$
$$(\{at(car1, Edinburgh)\}, \{at(car1, London)\}, \{at(car1, Edinburgh)\}),$$
$$(\{at(car2, London)\}, \{at(car2, Edinburgh)\}, \{at(car2, London)\}),$$
$$(\{at(car2, Edinburgh)\}, \{at(car2, London)\}, \{at(car2, Edinburgh)\})$$

Schematic Actions: Quantification

Example

$\exists x \in \{A, B, C\} : at(x, SB)$ is a short-hand for?

$at(A, SB) \lor at(B, SB) \lor at(C, SB)$.

Quantification in Formulas

Finite disjunctions $\varphi(o_1) \lor \cdots \lor \varphi(o_n)$ represented as $\exists x \in \{o_1, \ldots, o_n\} : \varphi(x)$.

Finite conjunctions $\varphi(o_1) \land \cdots \land \varphi(o_n)$ represented as $\forall x \in \{o_1, \ldots, o_n\} : \varphi(x)$.

Quantification over Effects

Finite list of conditional effects WHEN $\varphi(o_i)$ DO $\psi(o_i)$ represented as $\forall x \in \{o_1, \ldots, o_n\}$: WHEN $\varphi(o_i)$ DO $\psi(o_i)$.

Questionnaire

Question!

Is the grounding process polynomial in the size of its input?

(A): Yes (B): No

→ If an action schema has k parameters, and there are n objects each of these parameters can be instantiated with, then there are n^k grounded actions. Same for predicates. Grounding is exponential in operator and predicate arity.

- In practice, this is often ok, many domains have maximum arity 2 or 3.
- However, this is NOT always so! (E.g., natural language generation)
- Grounding typically leads to more efficient planning in the cases where it is feasible; in the other cases, lifted planning is needed.
- There has been little research on lifted planning in the last 2 decades.

PDDL Basics

The Planning Domain Definition Language (PDDL):

- Variants used by almost all implemented planning systems.
- Supports a formalism comparable to what we have outlined above (including schematic operators and quantification).
- Syntax inspired by the Lisp programming language: e.g., prefix notation for formulas

```
(and (or (on A B) (on A C))
     (or (on B A) (on B C))
     (or (on C A) (on A B)))
```

- The planner input is separated into a domain file (predicates, types, action schemas) and a problem file (objects, initial state, goal).

PDDL Domain Files

A PDDL domain file consists of:

1. `(define (domain <name>)`
2. A requirements definition (use ":adl :typing" by default).
3. Definitions of types (each object variable has a type).
4. Definitions of predicates.
5. Definitions of action schemas.

Domain File Types and Predicates: Example Blocksworld

```
(define (domain Blocksworld)
  (:requirements :adl :typing)
  (:types block - object
          blueblock smallblock - block)
  (:predicates (on ?x - smallblock ?y - block)
               (ontable ?x - block)
               (clear ?x - block))
```

Action Schema: Example Blocksworld

```
(:action fromtable
    :parameters (?x - smallblock ?y - block)
    :precondition (and (not (= ?x ?y))
                       (clear ?x)
                       (ontable ?x)
                       (clear ?y))
    :effect
      (and (not (ontable ?x))
           (not (clear ?y))
           (on ?x ?y)))
```

PDDL Grammar: Action Schema

- `(:action <name>`
- List of parameters:

  `(?x - type1 ?y - type2 ?z - type3)`
- The precondition is a formula:

```
<predicate>
(and <formula> ... <formula>)
(or <formula> ... <formula>)
(not <formula>)
(forall (?x1 - type1 ... ?xn - typen) <formula>)
(exists (?x1 - type1 ... ?xn - typen) <formula>)
```

PDDL Grammar: Action Schema, ctd.

- The effect is a combination of literals, conjunction, conditional effects, and quantification over effects:

```
<predicate>
(not <predicate>)
(and <effect> ... <effect>)
(when <formula> <effect>)
(forall (?x1 - type1 ... ?xn - typen) <effect>)
```

PDDL Problem Files

A PDDL problem file consists of:

1. `(define (problem <name>)`
2. `(:domain <name>)`
   – to which domain does this problem belong?
3. Definitions of objects belonging to each type.
4. Definition of the initial state (list of ground predicates initially true).
5. Definition of the goal (a formula like action preconditions).

Problem File: Example Blocksworld

```
(define (problem example)
  (:domain Blocksworld)
  (:objects a b c - smallblock)
            d e - block
            f - blueblock)
  (:init (clear a) (clear b) (clear c)
         (clear d) (clear e) (clear f)
         (ontable a) (ontable b) (ontable c)
         (ontable d) (ontable e) (ontable f))

  (:goal (and (on a d) (on b e) (on c f)))
)
```

Example Run of FF (Fast Forward) planner

If you execute "`../ff -o domain.pddl -f p-n3.pddl`" gives:

```
ff: found legal plan as follows
step    0: MOVE D1 D2 PEG3
        1: MOVE D2 D3 PEG2
        2: MOVE D1 PEG3 D2
        3: MOVE D3 PEG1 PEG3
        4: MOVE D1 D2 PEG1
        5: MOVE D2 PEG2 D3
        6: MOVE D1 PEG1 D2
0.00 seconds total time
```

PDDL History

The development of PDDL is mainly driven by the **International Planning Competition (IPC):**

- **1998:** PDDL [McDermott *et al.* (1998)]
  STRIPS and ADL.
- **2000:** "PDDL subset for the 2000 competition" [Bacchus (2000)]
  STRIPS and ADL.
- **2002:** PDDL2.1, Levels 1-3 [Fox and Long (2003)]
  Numeric and temporal planning.
- **2004:** PDDL2.2 [Hoffmann and Edelkamp (2005)]
  Derived predicates and timed initial literals.
- **2006:** PDDL3 [Gerevini *et al.* (2009)]
  Soft goals and trajectory constraints.

PDDL in 2002

Maria Fox and Derek Long promoted **numeric and temporal planning:**

- **PDDL2.1 level 1**: As in IPC'00.
- **PDDL2.1 level 2**: Level 1 plus numeric fluents. Comparisons between numeric expressions are allowed as logical atoms:
  `(>= (fuel) (* (dist ?x ?y) (consumption)))`
  Effects can modify fluents by numeric expressions:
  `(decrease (fuel) (* (dist ?x ?y) (consumption)))`
- **PDDL2.1 level 3**: Level 2 extended with action durations. Actions take an amount of time given by the value of a numeric expression:
  `(= ?duration (/ (dist ?x ?y) (speed))`
  Conditions/effects are applied at either start or end of action:
  `(at start (not (at ?x))) (at end (at ?y))`

PDDL in 2004

PDDL2.1 was (and is still today) considered a challenge, so only two relatively minor language extensions went into **PDDL2.2:**

- Derived predicates: Predicates that are not affected by the actions. Their value is instead derived via a set of derivation rules of the form **IF** $\varphi(\overline{x})$ **THEN** $P(\overline{x})$.
  Example: Flow of current in an electricity network.
  ```
  (:derived (fed ?x)
              (exists ?y (and (connected ?x ?y) (fed ?y))))
  ```
- Timed Initial Literals: Literals that will become true, independently of the actions taken, at a pre-specified point in time.
  Example: Opening/closing times.
  ```
  (at 9 (shop-open)) (at 18 (not (shop-open)))
  ```

PDDL in 2006

Actually, Gerevini & Long thought that PDDL2.2 is still not enough, and extended it with various complex constructs for expressing preferences over soft goals, as well as trajectory constraints, to obtain **PDDL3** . . .

. . . which I am not gonna describe here :-)

In 2008, Malte Helmert offered to introduce an FDR encoding as the front-end language.

Only few people wanted to invest the work of replacing their planner front-end, and the language ended up not being used. (Legacy system STRIPS, remember?)

PDDL for Planning under Uncertainty

There are numerous formalism variants, and numerous people made their own private PDDL extensions as needed for their work.
→ PDDL is less standardized for planning under uncertainty.

As used in the uncertainty tracks of the IPC:

- **2004, 2006, 2008**: Probabilistic PDDL (PPDDL) [Younes *et al.* (2005)]. Probability distributions over action effects:
```
(probabilistic 0.166 (dice-1)
               0.166 (dice-2) ... 0.17 (dice-6))
```
- **2006, 2008**: PPDDL with non-deterministic extension [Bonet and Givan (2006)]. Non-deterministic action effects:
```
(oneof (dice-1) (dice-2) ... (dice-6))
```
- **2011**: Relational Dynamic Influence Diagram Language (RDDL) [Sanner (2010)]. Describes probabilistic planning in terms of dynamic Bayesian networks ... [not considered here].

Summary

- PDDL is the de-facto standard for classical planning, as well as extensions to numeric/temporal planning, soft goals, trajectory constraints.
- PDDL is used in the International Planning Competition (IPC).
- PDDL uses a schematic encoding, with variables ranging over objects similarly as in predicate logic. Most implemented systems use grounding to transform this into a propositional encoding.
- PDDL has a Lisp-like syntax.

References I

Fahiem Bacchus. *Subset of PDDL for the AIPS2000 Planning Competition*. The AIPS-00 Planning Competition Comitee, 2000.

Blai Bonet and Robert Givan. 5th international planning competition: Non-deterministic track – call for participation. In *Proceedings of the 5th International Planning Competition (IPC'06)*, 2006.

Maria Fox and Derek Long. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research*, 20:61–124, 2003.

Alfonso Gerevini, Patrik Haslum, Derek Long, Alessandro Saetti, and Yannis Dimopoulos. Deterministic planning in the fifth international planning competition: PDDL3 and experimental evaluation of the planners. *Artificial Intelligence*, 173(5-6):619–668, 2009.

Jörg Hoffmann and Stefan Edelkamp. The deterministic part of ipc-4: An overview. *Journal of Artificial Intelligence Research*, 24:519–579, 2005.

Drew McDermott, Malik Ghallab, Adele Howe, Craig Knoblock, Ashwin Ram, Manuela Veloso, Daniel Weld, and David Wilkins. *The PDDL Planning Domain Definition Language*. The AIPS-98 Planning Competition Comitee, 1998.

References II

Scott Sanner. Relational dynamic influence diagram language (rddl): Language description. Available at `http://users.cecs.anu.edu.au/~ssanner/IPPC_2011/RDDL.pdf`, 2010.

Håkan L. S. Younes, Michael L. Littman, David Weissman, and John Asmuth. The first probabilistic track of the international planning competition. *Journal of Artificial Intelligence Research*, 24:851–887, 2005.