Constraint Programming & Planning

Agostino Dovier

Università di Udine
Dipartimento di Matematica e Informatica

Udine, DICEMBRE 2014

Competitions

There are three main competitions related to the topics of this course

1. **ASP competition**: organized in “odd” years, together with the conference LPNMR. Two main tracks: modeling (any dialect of a logic programming language is allowed) and solving (the models are given as ASP program as inputs for participating solvers)
2. **Minizinc challenge**: it is the competition for CP solvers. Models are written in Minizinc and solvers should be able to understand them. Organized yearly together with CP.
3. **International Planning Competition (IPC)**: organized every three years (2014, the last one) within the ICAPS conference. Models are written in PDDL.

Minizinc

- Minizinc is defined by NICTA
- You can download it from `http://www.minizinc.org/`
- You'll find a tutorial by Marriott and Stuckey there
- Typically a Minizinc model is first translated to Flatzinc using `mzn2fzn`
- A Flatzinc model is an *unfolded* version of the Minizinc one; basically it is a sequence of simple (flat) constraints
- Any modern constraint solver reads Flatzinc models as input

Syntax

- Variables (and parameters/constants) need to be typed. E.g.

  `par int:  a = 3;`

  `var int:  b;`

  Parameters should be assigned asap and are assigned once.
  `par` is the default value. `var` should be made explicit.
- Possible types for var/par are (plus string):

int	: integer variables (e.g. FD)
bool	: Boolean variables (particular cases of FD)
float	: floating point variables (for hybrid modeling)

- A variable should be assigned to a domain. E.g.,

  `var 0..100:v;` for intervals domain (typical case)

  `var {0,2,4,6}:w;` for explicitly listed domains

Syntax

You can define single/multi-dimensional arrays of variables:

- array [indexset1, indexset2, ...] of var type: varname;
- For instance:
  ```
  array [0..2] of var 1..5 :  v;
  array [1..5,1..5] of var 0..2 :  M;
  ```
- arrays are accessed as V[i], M[i,j].
- Set of integers as domains are allowed.
  ```
  set of 1..8 :  s;
  ```
  `s` is any subset of $\{1, \ldots, 8\}$. You can use membership (in), set inclusion (subset, superset), union (union), intersection (inter), set difference (diff), symmetric difference (symdiff) and cardinality (card) to build expressions with set variables.

Syntax

Constraints are added explicitly either in a flat or compact way. E.g.,

- constraint $a + b < 100$;
- constraint $a \backslash/ b$; (this means $a \vee b$ for Boolean variables)
- constraint alldifferent(V); (where V is an array of variables: the global constraint should be imported using `import`)
- constraint forall(EXPRESSION); (where EXPRESSION is a complex statement, such as a list comprehension). E.g.

  ```
  forall( [ v[ i ] != v[ j ] | i , j in 1..3 where i <
  j ] );
  ```

  (You should to read the manual for the syntax of EXPRESSIONs, of course)
- There is a simplified, user-friendly version:

  ```
  forall(i,j in 1..3 where i<j) (v[ i ] != v[ j ]);
  ```

Syntax

You can choose the search directive:

- solve satisfy;
- solve maximize(<Arithmetic EXPRESSION>);
- solve minimize(<Arithmetic EXPRESSION>);
- Example of expressions can be a single variable or a function.

n-Queens

```
include "alldifferent.mzn";

int: n = 8;
array [1..n] of var 1..n: queens;

constraint alldifferent(queens);
constraint forall(i,j in 1..n where i<j)
                  (j-i != abs(queens[i]-queens[j]));

solve satisfy;

output ["\n  Queens:", show(queens), "\n"];
```

n-Queens
with maximization example

```
include "alldifferent.mzn";

int: n = 8;
array [1..n] of var 1..n: queens;

constraint alldifferent(queens);
constraint forall(i,j in 1..n where i<j)
                  (j-i != abs(queens[i]-queens[j]));

solve maximize queens[1]+queens[2]+queens[3];

output ["\n  Queens:", show(queens), "\n"];
```

PDDL

- Planning Domain Definition Language (PDDL) is a “standard” action description language introduced in 1988 by a group of top researchers in AI (Ghallab, Howe, Knoblock, Drew McDermott, Ram, Veloso, Weld, Wilkins).
- The programming style is *functional* (declarative, but not logic programming)
- There is a tradition of functional programming within the “hard core” of AI and Planning due to Mc Carthy school.

PDDL

- Planning Domain Definition Language (PDDL) is a “standard” action description language introduced in 1988 by a group of top researchers in AI (Ghallab, Howe, Knoblock, Drew McDermott, Ram, Veloso, Weld, Wilkins).
- The programming style is *functional* (declarative, but not logic programming)
- There is a tradition of functional programming within the “hard core” of AI and Planning due to Mc Carthy school.

PDDL: domain definition

```
(define (domain DOMAIN_NAME)
  (:requirements [:strips] [:equality] [:typing] [:adl])
  (:predicates
              (PREDICATE_1_NAME ?A1 ?A2 ... ?AN)
               (PREDICATE_2_NAME ?A1 ?A2 ... ?AN)
       ...
  )

  (:action ACTION_1_NAME
    [:parameters (?P1 ?P2 ... ?PN)]
    [:precondition PRECOND_FORMULA]
    [:effect EFFECT_FORMULA]
   )

  (:action ACTION_2_NAME
    ...)
)
```

PDDL: formulas

- For STRIPS domains, a precondition formula may be: an atomic formula `(PREDICATE_NAME ARG1 ... ARGN)` or a conjunction of atomic formulas: `(and ATOM1 ... ATOMN)`
- For ADL domains, a precondition may in addition be: A general negation, conjunction or disjunction:
  `(not CONDITION_FORMULA) (and CONDITION_FORMULA1 ... CONDITION_FORMULAN) (or CONDITION_FORMULA1 ... CONDITION_FORMULAN)`
  A quantified formula:
  `(forall (?V1 ?V2 ...) CONDITION_FORMULA)`

PDDL: formulas

- For STRIPS domains, an effect formula may be: An added atom: `(PREDICATE_NAME ARG1 ... ARGN)` The predicate arguments must be parameters of the action (or constants declared in the domain, if the domain has constants). A deleted atom: `(not (PREDICATE_NAME ARG1 ... ARGN))` A conjunction of atomic effects: `(and ATOM1 ... ATOMN)`
- For ADL domains an effect formula may in addition contain: A conditional effect: `(when CONDITION_FORMULA EFFECT_FORMULA)` or a universally quantified formula: `(forall (?V1 ?V2 ...) EFFECT_FORMULA)`

PDDL: Problem Definition

The problem definition contains the objects present in the problem instance, the initial state description and the goal:

```
(define (problem PROBLEM_NAME)
  (:domain DOMAIN_NAME)
  (:objects OBJ1 OBJ2 ... OBJ_N)
  (:init ATOM1 ATOM2 ... ATOM_N)
  (:goal CONDITION_FORMULA)
  )
```

Other options such as action/plan cost can be set.

PDDL: Example

```
(:objects rooma roomb  ball1 ball2 ball3 ball4  left right)

(:predicates (ROOM ?x) (BALL ?x) (GRIPPER ?x) (at-robby ?x)
        (at-ball ?x ?y) (free ?x) (carry ?x ?y))

(:init (ROOM rooma) (ROOM roomb) (at-robby rooma)
    (BALL ball1) (BALL ball2) (BALL ball3) (BALL ball4)
    (GRIPPER left) (GRIPPER right) (free left) (free right)
    (at-ball ball1 rooma) (at-ball ball2 rooma)
    (at-ball ball3 rooma) (at-ball ball4 rooma))

(:goal (and (at-ball ball1 roomb) (at-ball ball2 roomb)
            (at-ball ball3 roomb) (at-ball ball4 roomb)))

(:action pick-up :parameters (?x ?y ?z)
   :precondition (and (BALL ?x) (ROOM ?y) (GRIPPER ?z)
           (at-ball ?x ?y) (at-robby ?y) (free ?z))
   :effect       (and (carry ?z ?x)
            (not (at-ball ?x ?y)) (not (free ?z))))
```