Planning as Model Checking

Manual & Assignments

Piergiorgio Bertoli   Marco Pistore   Marco Roveri

`[bertoli,pistore,roveri]@irst.itc.it`

ITC-IRST, Via Sommarive 18, 38055 Povo, Trento, Italy

Manual: plan synthesis

- Plan synthesis:

```
MBP-solve  [-plan_output <plan>]
           [-plan_validate]
           [-plan_simulate]
           <nupddl-files>
```

  where:

  - `-plan_output <plan>` enables the emission of the computed plan into file `<plan>` ("`-`" means stdout).
  - `-plan_validate` enables the validation of the computed plan.
  - `-plan_simulate` enables the simulation of the computed plan.
  - `<nupddl-files>` is a list of input nuPDDL files, describing a domain and a problem.
  - For advanced search options, see `MBP-solve -h`.

Manual: validation & simulation

- Plan validation:

  `MBP-validate  <nupddl-files>`

- Plan/Domain simulation:

  `MBP-simulate  [-domain] [-random] <nupddl-files>`

  where:

  - `-domain` enables the simulation of the domain in isolation.
  - `-random` enters random simulation mode.
  - `<nupddl-files>` is a list of input nuPDDL files, describing a domain, a problem and a plan.

ITC irst

NuPDDL: domain/problem language

- Backward PDDL compatibility:
  - Retains closed world assumption, inertiality, parametricity.
  - Includes most of PDDL up ADL layer.
  - Includes PDDL2.1 “functions” extension.
- No layered structure.
- Typing enforced.
- Allows nested quantifications and conditionals.
- Extension: Nondeterminism (initial, action effects).
- Extension: Partial observability.
- Extension: Goal classes.

nuPDDL: plan language

Overview:

- Consistently with theory, allows defining an automata.
- Simple plan structures easily captured.
- Syntax style taken from domain definition part of nuPDDL.
- User-friendly imperative-style constructs supported.

nuPDDL: plan language (II)

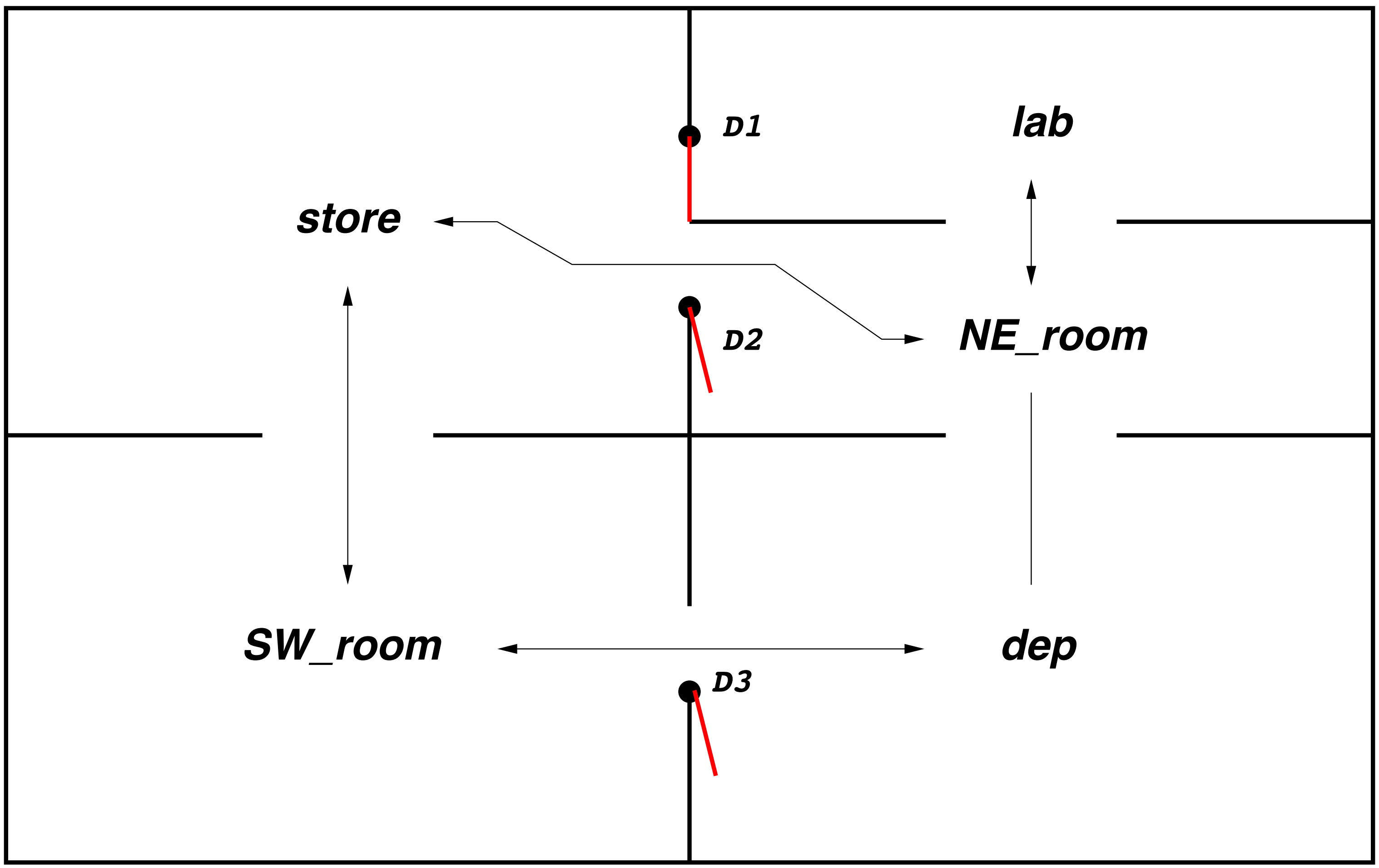
- A plan *may* feature a set of typed `:planvars`.
- Plan vars are `:initialized` (otherwise they assume a default).
- Basic plan steps:
  - `(done)` signals ending of plan.
  - `(fail)` signals plan failure.
  - `(evolve (assign (next(v1) val1)) ... (action (act)))`
  - ..or simply: `(action (act))`
- Steps can be `sequenced`.
  - Branch constructs:
  - `(if (cond) plan1 plan2)`
  - `(switch (case (cond1) plan1) ... (else plan_else))`
- Imperative-style constructs: `label` and `goto`
- Iterations: `while` and `repeat`

NuPDDL: CTL goals

Do Reach p ("strong goal"):	`(af p)`
Try Reach p ("weak goal"):	`(ef p)`
Keep Trying Reach p ("strong cyclic goal"):	`(aw (ef p) p)`
Continuously Try Reach p:	`(ag (ef p))`
Do Maintain p:	`(ag p)`
Try Maintain p:	`(eg p)`
Do Maintain p Until q:	`(au p q)`
In All Next States p:	`(ax p)`
In Some Next States p:	`(ex p)`
And:	(and g_1 g_2 g_3...)
Or:	(or g_1 g_2 g_3...)
Implies:	`(imply p g)`

ITC irst

Starting example: robot navigation



A nuPDDL domain model: `robot.npddl`

```
(define (domain robot_navigation)
  (:types room)
  (:constants store lab NE_room SW_room dep - room)
  (:functions  (robot_position) - room)

  (:action move_robot_up
     :precondition (or (= (robot_position) SW_room )
                       (= (robot_position) dep     )
                       (= (robot_position) NE_room ))
     :effect (and
                 (when (= (robot_position) SW_room ) (assign (robot_position) store   ))
                 (when (= (robot_position) dep     ) (assign (robot_position) NE_room ))
                 (when (= (robot_position) NE_room ) (assign (robot_position) lab     ))))

  (:action move_robot_down
     :precondition (or (= (robot_position) store   )
                       (= (robot_position) lab     )
                       (= (robot_position) NE_room ))
     :effect (and
                 (when (= (robot_position) store   ) (assign (robot_position) SW_room ))
                 (when (= (robot_position) lab     ) (assign (robot_position) NE_room ))
                 (when (= (robot_position) NE_room ) (assign (robot_position) dep     ))))

  (:action move_robot_right
     :precondition (or (= (robot_position) SW_room )
                       (= (robot_position) store   ))
     :effect (and
                 (when (= (robot_position) SW_room ) (assign (robot_position) dep))
                 (when (= (robot_position) store   ) (assign (robot_position) NE_room))))

  (:action move_robot_left
     :precondition (or (= (robot_position) dep     )
                       (= (robot_position) NE_room ))
     :effect (and
                 (when (= (robot_position) dep     ) (assign (robot_position) SW_room ))
                 (when (= (robot_position) NE_room ) (assign (robot_position) store   )))))
```

A silly nonsensical plan

```
(define (plan silly_plan)
        (:domain robot_navigation)
        (:problem navigation_problem)
        (:planvars visited_lab - boolean
                   visited_SW_room_no - (range 0 10))
        (:init
            (= (visited_SW_room_no) 0)
            (= (visited_lab) 0)
        )
        (:body
            (sequence
                (while (< (visited_lab) 10)
                    (sequence
                       (evolve (assign
                                  (next (visited_SW_room_no))
                                  (+ (visited_SW_room_no) 1))
                               (action (move_robot_down)))
                       (action (move_robot_up))))
                (action (move_robot_right))
                (label i_am_in_lab (if (= (robot_position) lab)
                    (sequence
                      (evolve
                              (assign (next(visited_lab)) 1)
                              (action (move_robot_down)))
                      (action (move_robot_down)))
                    (action (move_robot_down))))
                (switch
                   (case (= (robot_position) dep) (done))
                   (case (= (robot_position) lab) (goto i_am_in_lab))
                   (case (= (robot_position) store) (fail))
                   (else (fail))))))
```

Questions

1. Synthesize (and save) a strong plan for reaching *dep* from *store*.
2. Synthesize (and save) a conformant plan for the same problem.

Now suppose D3 is closed (initially and forever):

- Update the domain.
- Check whether the plans generated before are still valid.
- Synthesize:
  - A strong plan for reaching *dep* from *store*.
  - A conformant plan for the same problem.

ITC irst

A nondeterministic domain

Now suppose that:

1. (initially and forever) every door is open.
2. going east from *store* may lead to either *lab* or *NE_room*.

Then:

- Update the domain. *(Tip: modeling doors is not necessary...)*
- Synthesize a strong plan to <u>reach *lab* from *store*</u>.
- Synthesize a conformant plan for the same problem.
- Is the strong plan valid assuming no observability?

ITC irst

Now:

- D3 is uncontrollable, D1 and D2 are open.
- The robot “bounces” on D3 if closed.

Then:

- Update the domain. *(Tip: one can model D3 through the way it affects movements...)*
- Synthesize a strong plan to go from *store* to *dep*.
- Synthesize a conformant plan for the same problem.

Extended goals

With D3 uncontrollable, D1 and D2 open, suppose *lab* is a dangerous room.

1. Is there a strong plan from *store* to *dep*, admitting passage into *lab*?
2. Is there a strong plan that leads from *store* to *dep* avoiding *lab*?
3. Is there a weak plan for the same problem?
4. Is there a strong cyclic plan for the same problem?

ITC irst

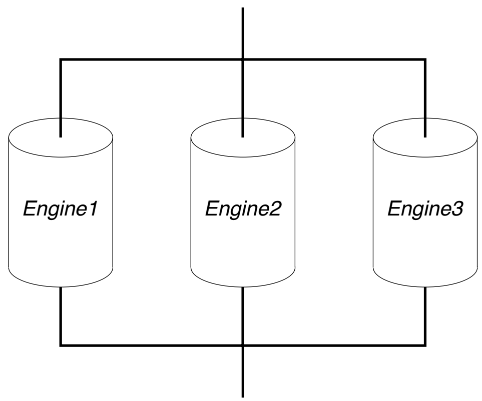
Problems with Partial Observability

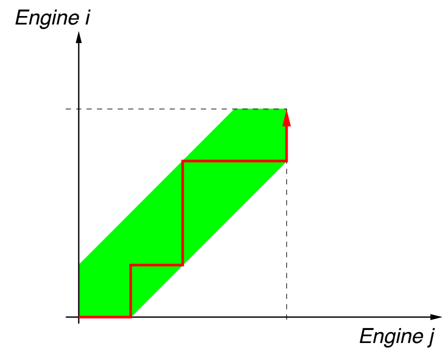
Now suppose that:

- Exactly one of the doors is open.
- The robot cannot try traversing a locked door.
- The robot can sense whether it can move in one direction or not.

- Is there a strong plan <u>from store to dep</u>? A weak plan?
- Is there a conformant plan? A strong plan using observations?
- Suppose the robot can smell whether it's in the *lab*. Is there a strong plan using observations?

Advanced assignments: extended goals (I)





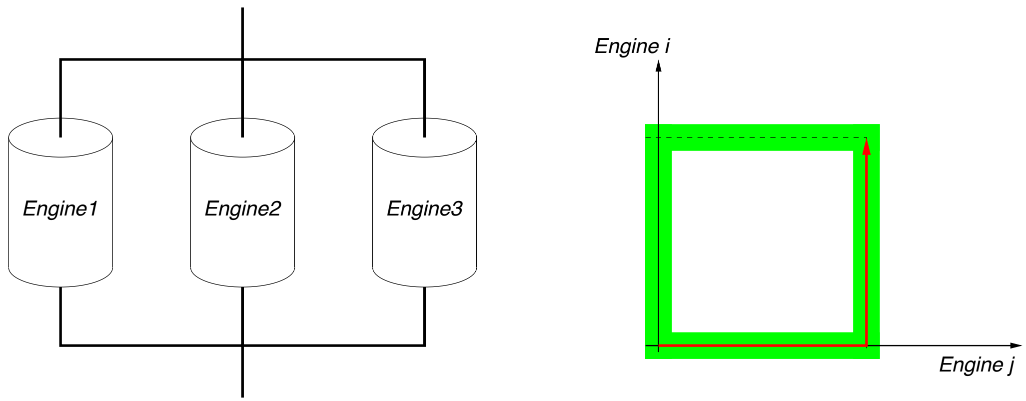
- Three engines, each providing from 0 to 4 levels of power.
- Engines start from being off.
- Problem: reach maximum power while keeping balancing (see figure).

Tasks:

1. Model the domain.
2. Synthesize a strong plan for the problem (if there is one).
3. Simulate the plan.
4. Write a smarter plan, validate and simulate it.

ITC irst

Advanced assignments: extended goals (II



- Now problem is: reach maximum power following saturation policy.
- Saturation: at most one engine is “half way through” (see figure).

Tasks:

1. Model the domain.
2. Synthesize a strong plan for the problem.
3. Write a smarter plan and validate it.

ITC irst

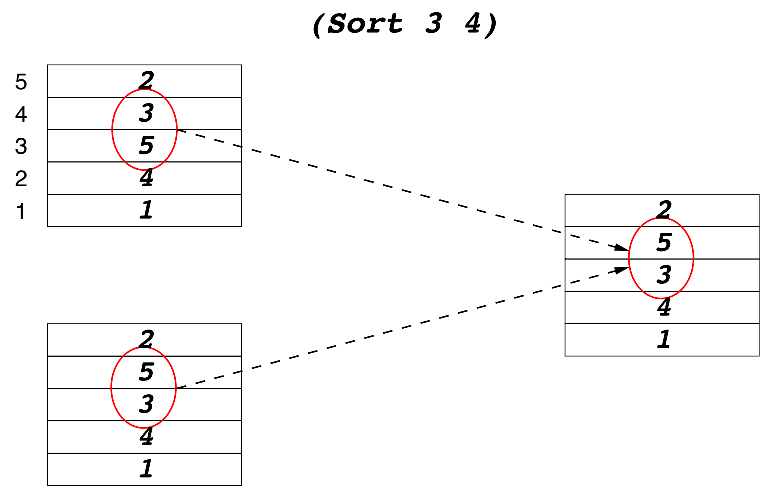
Advanced assignments: extended goals (III

Possible advanced goals:

- Changing request level:
  - the sum of the power provided by the engines should reach a given *request level*;
  - the request level may increase or decrease.
- Alarm:
  - whenever an alarm is raised, all the engines should be turned off;
  - the alarm ends once all the engines are off.
- Switching policy:
  - a request of switch from saturation to balancing or vice-versa can be raised at run-time.

ITC
irst

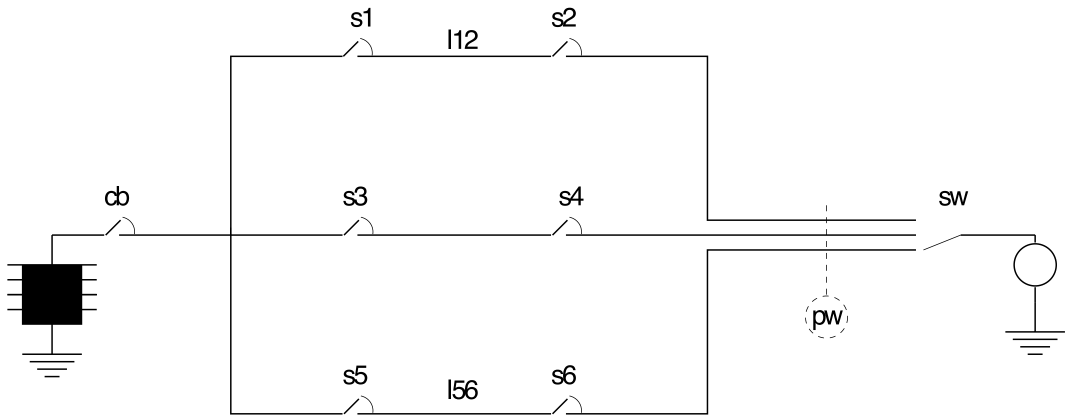
Advanced assignments: conformant (I)



- A stack of 5 numbers, each ranging from 1 to 5.
- An atomic pairwise *(sort x y)* operation.
- Any configuration possible at start.
- The stack must be sorted at the end.

Tasks: Model the domain and find a conformant plan for the problem.

Advanced assignments: PO (I)



- An electric circuit. Possible actions: open/close *cb*,*s1*,...,*s6*.
- One of *l12* or *l56* has a shortcircuit.
- When *cb* feeds a shortcircuit it automatically reopens.
- Switch *s3* is unreliable: it may not obey.
- Sensor *pw* tells whether power gets to 3-position switch *sw*.
- Goal: turn on light. Initially *cb*,*s1*,...,*s6* are open, *sw* is at position 1.
- Task: model the domain, solve the problem,.
- Task: design a smarter plan, validate and simulate it.

ITC irst