Introduction to Artificial Intelligence

English practicals 5: Automated Planning

Marika Ivanová

Department of Theoretical Computer Science and Mathematical Logic (KTIML)
Faculty of Mathematics and Physics

March 15th 2021

A small reminder

- Representation of information changing with time: using time-annotated propositional variables (fluents)
- **Observation model:** connects observation with information in the world model
- **Transition model:** describes evolution of world after applying actions, e.g. using effect axioms:

$$L^t_{x,y} \wedge FacingEast^t \wedge Forward^t \Rightarrow (L^{t+1}_{x+1,y} \wedge \neg L^{t+1}_{x,y})$$

A small reminder

- Representation of information changing with time: using time-annotated propositional variables (fluents)
- **Observation model:** connects observation with information in the world model
- **Transition model:** describes evolution of world after applying actions, e.g. using effect axioms:

$$L^t_{x,y} \wedge FacingEast^t \wedge Forward^t \Rightarrow (L^{t+1}_{x+1,y} \wedge \neg L^{t+1}_{x,y})$$

function SATPLAN(*init*, *transition*, *goal*, T_{max}) **returns** solution or failure
  inputs: *init*, *transition*, *goal*, constitute a description of the problem
    T_{max}, an upper limit for plan length

  for $t = 0$ **to** T_{max} **do**
    cnf ← TRANSLATE-TO-SAT(*init*, *transition*, *goal*, t)
    model ← SAT-SOLVER(*cnf*)
    if *model* is not null **then**
      return EXTRACT-SOLUTION(*model*)
  return *failure*

A small reminder

- Representation of information changing with time: using time-annotated propositional variables (fluents)
- **Observation model:** connects observation with information in the world model
- **Transition model:** describes evolution of world after applying actions, e.g. using effect axioms:

$$L^t_{x,y} \wedge FacingEast^t \wedge Forward^t \Rightarrow (L^{t+1}_{x+1,y} \wedge \neg L^{t+1}_{x,y})$$

```
function SATPLAN(init, transition, goal, T_max) returns solution or failure
  inputs: init, transition, goal, constitute a description of the problem
          T_max, an upper limit for plan length

  for t = 0 to T_max do
    cnf ← TRANSLATE-TO-SAT(init, transition, goal, t)
    model ← SAT-SOLVER(cnf)
    if model is not null then
      return EXTRACT-SOLUTION(model)
  return failure
```

- Enormous size of CNFs → **Situation calculus** (first order logic)

A small reminder

Classical planning

- **State** s is a set of instantiated atoms (without variables)
  - fluents (changing in states): $At(r_1, l_2)$
  - rigid atoms (does not change in states): $Adjacent(l_1, l_2)$
- **Goal** g is a set of instantiated literals (may contain variables - $At(p, PRG) \wedge Plane(p)$)

  a state s satisfies the goal condition g iff $g^+ \subseteq s \wedge g^- \cap s = \emptyset$
- **Action schema (operator)**: represents a set of ground actions

  $Action(Fly(p, from, to),$

  $PRECOND : At(p, from) \wedge Plane(p) \wedge Airport(from) \wedge Airport(to)$
  $EFFECT : \neg At(p, from) \wedge At(p, to)$
- **Action** is a fully instantiated operator

Selected quiz questions

Selected quiz questions

- What is a frame problem? How is it solved?

Selected quiz questions

- What is a frame problem? How is it solved?

  Effect axioms do not say anything about what is not changed by an action

Selected quiz questions

- What is a frame problem? How is it solved?

  Effect axioms do not say anything about what is not changed by an action

  Frame axioms: explicitly assert all the propositions that remain unchanged - axioms about actions, too many

Selected quiz questions

- What is a frame problem? How is it solved?

  Effect axioms do not say anything about what is not changed by an action

  Frame axioms: explicitly assert all the propositions that remain unchanged - axioms about actions, too many

  Successor-state axioms: about fluents: define the truth value of a fluent at time $t+1$ in terms of fluents and actions at time t

  $$F^{t+1} \Leftrightarrow ActionCausesF^{t} \lor (F^{t} \land \neg ActionCausesNotF^{t})$$

  $$HaveArrow^{t+1} \Leftrightarrow (HaveArrow^{t} \land \neg Shoot^{t})$$

Selected quiz questions

- What is a frame problem? How is it solved?

  Effect axioms do not say anything about what is not changed by an action

  Frame axioms: explicitly assert all the propositions that remain unchanged - axioms about actions, too many

  Successor-state axioms: about fluents: define the truth value of a fluent at time $t + 1$ in terms of fluents and actions at time t

  $$F^{t+1} \Leftrightarrow ActionCausesF^t \vee (F^t \wedge \neg ActionCausesNotF^t)$$

  $$HaveArrow^{t+1} \Leftrightarrow (HaveArrow^t \wedge \neg Shoot^t)$$

- Why do we need to set some propositional variables to false in the description of the initial state for SATPlan?

Selected quiz questions

- What is a frame problem? How is it solved?

  Effect axioms do not say anything about what is not changed by an action

  Frame axioms: explicitly assert all the propositions that remain unchanged - axioms about actions, too many

  Successor-state axioms: about fluents: define the truth value of a fluent at time $t+1$ in terms of fluents and actions at time t

  $$\mathbf{F^{t+1} \Leftrightarrow ActionCausesF^{t} \lor (F^{t} \land \neg\, ActionCausesNotF^{t})}$$

  $$\mathbf{HaveArrow^{t+1} \Leftrightarrow (HaveArrow^{t} \land \neg\, Shoot^{t})}$$

- Why do we need to set some propositional variables to false in the description of the initial state for SATPlan?

  We must specify not only what is true, but also what is false, because if some variables of the initial state were unassigned, the solver could assign them incositently

Selected quiz questions

- What is a frame problem? How is it solved?

  Effect axioms do not say anything about what is not changed by an action

  Frame axioms: explicitly assert all the propositions that remain unchanged - axioms about actions, too many

  Successor-state axioms: about fluents: define the truth value of a fluent at time $t + 1$ in terms of fluents and actions at time t

  $$\mathbf{F^{t+1} \Leftrightarrow ActionCausesF^{t} \vee (F^{t} \wedge \neg\, ActionCausesNotF^{t})}$$

  $$\mathbf{HaveArrow^{t+1} \Leftrightarrow (HaveArrow^{t} \wedge \neg\, Shoot^{t})}$$

- Why do we need to set some propositional variables to false in the description of the initial state for SATPlan?

  We must specify not only what is true, but also what is false, because if some variables of the initial state were unassigned, the solver could assign them incosistently

- How does SATPlan find the length of the plan? Does it always guarantee to find the shortest plan?

Selected quiz questions

- What is a frame problem? How is it solved?

  Effect axioms do not say anything about what is not changed by an action

  Frame axioms: explicitly assert all the propositions that remain unchanged - axioms about actions, too many

  Successor-state axioms: about fluents: define the truth value of a fluent at time $t+1$ in terms of fluents and actions at time t

  $$\mathbf{F^{t+1} \Leftrightarrow ActionCausesF^{t} \vee (F^{t} \wedge \neg ActionCausesNotF^{t})}$$

  $$\mathbf{HaveArrow^{t+1} \Leftrightarrow (HaveArrow^{t} \wedge \neg Shoot^{t})}$$

- Why do we need to set some propositional variables to false in the description of the initial state for SATPlan?

  We must specify not only what is true, but also what is false, because if some variables of the initial state were unassigned, the solver could assign them incosistently

- How does SATPlan find the length of the plan? Does it always guarantee to find the shortest plan?

  By an iterative increase of the maximum time - always finds a shortest plan

Selected quiz questions

Selected quiz questions

- What is the role of action exclusion axioms?

Selected quiz questions

- What is the role of action exclusion axioms?

  Elimination of parallel executions of certain actions

Selected quiz questions

- What is the role of action exclusion axioms?

  Elimination of parallel executions of certain actions

- Describe formally when two actions interfere using the classical planning model.

Selected quiz questions

- What is the role of action exclusion axioms?

  Elimination of parallel executions of certain actions

- Describe formally when two actions interfere using the classical planning model.

  $effect^{+}(a_1) \cap effect^{-}(a_2) = \emptyset$

  $effect^{+}(a_2) \cap effect^{-}(a_1) = \emptyset$

Selected quiz questions

- What is the role of action exclusion axioms?

  Elimination of parallel executions of certain actions

- Describe formally when two actions interfere using the classical planning model.

  $effect^{+}(a_1) \cap effect^{-}(a_2) = \emptyset$

  $effect^{+}(a_2) \cap effect^{-}(a_1) = \emptyset$

  $precond^{+}(a_1) \cap effect^{-}(a_2) = \emptyset$

  $precond^{-}(a_1) \cap effect^{+}(a_2) = \emptyset$

Selected quiz questions

- What is the role of action exclusion axioms?

  Elimination of parallel executions of certain actions

- Describe formally when two actions interfere using the classical planning model.

  $effect^{+}(a_1) \cap effect^{-}(a_2) = \emptyset$

  $effect^{+}(a_2) \cap effect^{-}(a_1) = \emptyset$

  $precond^{+}(a_1) \cap effect^{-}(a_2) = \emptyset$

  $precond^{-}(a_1) \cap effect^{+}(a_2) = \emptyset$

  and similarly for a_2 and a_1

Selected quiz questions

- What is the role of action exclusion axioms?

  Elimination of parallel executions of certain actions
- Describe formally when two actions interfere using the classical planning model.

  $effect^+(a_1) \cap effect^-(a_2) = \emptyset$

  $effect^+(a_2) \cap effect^-(a_1) = \emptyset$

  $precond^+(a_1) \cap effect^-(a_2) = \emptyset$

  $precond^-(a_1) \cap effect^+(a_2) = \emptyset$

  and similarly for a_2 and a_1
- Can action, that is not relevant for the goal, be used as the last action in the solution plan? If yes, what can we say about such a solution plan?

Selected quiz questions

- What is the role of action exclusion axioms?

  Elimination of parallel executions of certain actions

- Describe formally when two actions interfere using the classical planning model.

  $effect^+(a_1) \cap effect^-(a_2) = \emptyset$

  $effect^+(a_2) \cap effect^-(a_1) = \emptyset$

  $precond^+(a_1) \cap effect^-(a_2) = \emptyset$

  $precond^-(a_1) \cap effect^+(a_2) = \emptyset$

  and similarly for a_2 and a_1

- Can action, that is not relevant for the goal, be used as the last action in the solution plan? If yes, what can we say about such a solution plan?

  Yes, but the plan is not the shortest possible.

Selected quiz questions

- What is the role of action exclusion axioms?

  Elimination of parallel executions of certain actions

- Describe formally when two actions interfere using the classical planning model.

  $effect^{+}(a_1) \cap effect^{-}(a_2) = \emptyset$

  $effect^{+}(a_2) \cap effect^{-}(a_1) = \emptyset$

  $precond^{+}(a_1) \cap effect^{-}(a_2) = \emptyset$

  $precond^{-}(a_1) \cap effect^{+}(a_2) = \emptyset$

  and similarly for a_2 and a_1

- Can action, that is not relevant for the goal, be used as the last action in the solution plan? If yes, what can we say about such a solution plan?

  Yes, but the plan is not the shortest possible.

- What is the relation between goal (goal condition) and goal state?

Selected quiz questions

- What is the role of action exclusion axioms?

  Elimination of parallel executions of certain actions

- Describe formally when two actions interfere using the classical planning model.

  $effect^{+}(a_1) \cap effect^{-}(a_2) = \emptyset$

  $effect^{+}(a_2) \cap effect^{-}(a_1) = \emptyset$

  $precond^{+}(a_1) \cap effect^{-}(a_2) = \emptyset$

  $precond^{-}(a_1) \cap effect^{+}(a_2) = \emptyset$

  and similarly for a_2 and a_1

- Can action, that is not relevant for the goal, be used as the last action in the solution plan? If yes, what can we say about such a solution plan?

  Yes, but the plan is not the shortest possible.

- What is the relation between goal (goal condition) and goal state?

  a goal is a set of instantiated literals - may contain variables

  a goal state is a state, a set of instantiated atoms - no variables

Selected quiz questions

- What is the role of action exclusion axioms?

  Elimination of parallel executions of certain actions

- Describe formally when two actions interfere using the classical planning model.

  $effect^+(a_1) \cap effect^-(a_2) = \emptyset$

  $effect^+(a_2) \cap effect^-(a_1) = \emptyset$

  $precond^+(a_1) \cap effect^-(a_2) = \emptyset$

  $precond^-(a_1) \cap effect^+(a_2) = \emptyset$

  and similarly for a_2 and a_1

- Can action, that is not relevant for the goal, be used as the last action in the solution plan? If yes, what can we say about such a solution plan?

  Yes, but the plan is not the shortest possible.

- What is the relation between goal (goal condition) and goal state?

  a goal is a set of instantiated literals - may contain variables

  a goal state is a state, a set of instantiated atoms - no variables

  Thus, several goal states can satisfy one goal condition

Planning Definition Domain Language (PDDL)

Components of a PDDL planning task

Planning Definition Domain Language (PDDL)

Components of a PDDL planning task

- **Objects:** relevant things that appear in the world

Planning Definition Domain Language (PDDL)

Components of a PDDL planning task

- **Objects:** relevant things that appear in the world
- **Predicates:** relevant properties of objects, true/false

Planning Definition Domain Language (PDDL)

Components of a PDDL planning task

- **Objects:** relevant things that appear in the world
- **Predicates:** relevant properties of objects, true/false
- **Initial state:** state in which it starts

Planning Definition Domain Language (PDDL)

Components of a PDDL planning task

- **Objects:** relevant things that appear in the world
- **Predicates:** relevant properties of objects, true/false
- **Initial state:** state in which it starts
- **Goal specification:** what needs to be true

Planning Definition Domain Language (PDDL)

Components of a PDDL planning task

- **Objects:** relevant things that appear in the world
- **Predicates:** relevant properties of objects, true/false
- **Initial state:** state in which it starts
- **Goal specification:** what needs to be true
- **Actions/operators:** ways of changing the state

Planning Definition Domain Language (PDDL)

Components of a PDDL planning task

- **Objects:** relevant things that appear in the world
- **Predicates:** relevant properties of objects, true/false
- **Initial state:** state in which it starts
- **Goal specification:** what needs to be true
- **Actions/operators:** ways of changing the state

- PDDL is a language for standard planning in AI
- Model defined in two files:
  - Problem: list of objects, initial and goal state
  - Domain: list of variables and actions

PDDL: Template of a problem file

```
(define (problem <problem name>)
    (:domain <domain name>)
    <PDDL code for objects>
    <PDDL code for initial state>
    <PDDL code for goal specification>
)
```

PDDL: Template of a domain file

```
(define (domain <domain name>)
   <PDDL code for predicates>
   <PDDL code for first action>
   [...]
   <PDDL code for last action>
)
```

Example: Gripper 1/6

There are four balls and a robot Robby that can move between two rooms and pick up or drop a ball with either of his two arms. Initially, Robby and all balls are in the first room. We would like to move all the balls to the second room.

Example: Gripper 1/6

There are four balls and a robot Robby that can move between two rooms and pick up or drop a ball with either of his two arms. Initially, Robby and all balls are in the first room. We would like to move all the balls to the second room.

Objects

- 2 rooms `rooma, roomb`
- 4 balls `ball1, ball2, ball3, ball4`
- Robby's two arms `left, right`

Example: Gripper 1/6

There are four balls and a robot Robby that can move between two rooms and pick up or drop a ball with either of his two arms. Initially, Robby and all balls are in the first room. We would like to move all the balls to the second room.

Objects

- 2 rooms `rooma, roomb`
- 4 balls `ball1, ball2, ball3, ball4`
- Robby's two arms `left, right`

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

Example: Gripper 2/6

Predicates

- `ROOM(x)`: true iff x is a room
- `BALL(x)`: true iff x is a ball
- `GRIPPER(x)`: true iff x is one of the robot's arms

Example: Gripper 2/6

Predicates

- `ROOM(x)`: true iff x is a room
- `BALL(x)`: true iff x is a ball
- `GRIPPER(x)`: true iff x is one of the robot's arms
- `at-robby(x)`: true iff x is a room and robot is in x
- `at-ball(x,y)`: true iff x is a ball, y is a room and x is in y
- `free(x)`: true iff x is a robot's arm and x does not hold any ball

Example: Gripper 2/6

Predicates

- `ROOM(x)`: true iff x is a room
- `BALL(x)`: true iff x is a ball
- `GRIPPER(x)`: true iff x is one of the robot's arms
- `at-robby(x)`: true iff x is a room and robot is in x
- `at-ball(x,y)`: true iff x is a ball, y is a room and x is in y
- `free(x)`: true iff x is a robot's arm and x does not hold any ball
- `carry(x,y)`: true iff x is a robot's arm, y is a ball and x holds y

Example: Gripper 2/6

Predicates

- `ROOM(x)`: true iff x is a room
- `BALL(x)`: true iff x is a ball
- `GRIPPER(x)`: true iff x is one of the robot's arms
- `at-robby(x)`: true iff x is a room and robot is in x
- `at-ball(x,y)`: true iff x is a ball, y is a room and x is in y
- `free(x)`: true iff x is a robot's arm and x does not hold any ball
- `carry(x,y)`: true iff x is a robot's arm, y is a ball and x holds y

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

Example: Gripper 3/6

Initial state

Predicates that are true

- `ROOM(rooma) ROOM(roomb)`
- `BALL(ball1), BALL(ball2), BALL(ball3), BALL(ball4)`
- `GRIPPER(left), GRIPPER(right)`
- `at-robby(rooma)`
- `at-ball(ball1,rooma),..., at-ball(ball4, rooma)`

Everything else is false

Example: Gripper 3/6

Initial state

Predicates that are true

- ROOM(rooma) ROOM(roomb)
- BALL(ball1), BALL(ball2), BALL(ball3), BALL(ball4)
- GRIPPER(left), GRIPPER(right)
- at-robby(rooma)
- at-ball(ball1,rooma),..., at-ball(ball4, rooma)

Everything else is false

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

Example: Gripper 4/6

Action: robot movement

- Robot moves from room x to room y
- Preconditions: x and y must be rooms, Robby must be at x
- Effects: Robby is no longer at x, but is at y

Example: Gripper 4/6

Action: robot movement

- Robot moves from room x to room y
- Preconditions: x and y must be rooms, Robby must be at x
- Effects: Robby is no longer at x, but is at y

```
(:action move :parameters (?x ?y)
   :precondition (and (ROOM ?x) (ROOM ?y)
                      (at-robby ?x))
   :effect        (and (at-robby ?y)
                      (not (at-robby ?x))))
```

Example: Gripper 5/6

Action: pick-up

- Robot picks up ball x in room y with arms z
- Preconditions: x must be a ball, y must a room, x must be at y, Robby must be located at y and z is its arm, which must be empty
- Effects: Arms z carries x, x is no longer at y and z is no longer free

Example: Gripper 5/6

Action: pick-up

- Robot picks up ball x in room y with arms z
- Preconditions: x must be a ball, y must a room, x must be at y, Robby must be located at y and z is its arm, which must be empty
- Effects: Arms z carries x, x is no longer at y and z is no longer free

```
(: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))))
```

Example: Gripper 6/6

Action: drop

- Robby drops ball x at room y from arm z
- Preconditions: x bust be a ball, y must a room, and z must be an arm. Robby must be located at y and hold x in its arm z
- Effects: z no longer carries x, which is now at y, z becomes free

Example: Gripper 6/6

Action: drop

- Robby drops ball x at room y from arm z
- Preconditions: x bust be a ball, y must a room, and z must be an arm. Robby must be located at y and hold x in its arm z
- Effects: z no longer carries x, which is now at y, z becomes free

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

Exercise

The flat tire problem

Problem of changing flat tire: Initially, we have a flat tire on the axle and a good spare tire in the trunk. We want to have a good spare tire on the axle.

- Objects: Flat, Spare, Axle, Trunk
- Init: $Tire(Flat) \wedge Tire(Spare) \wedge At(Flat, Axle) \wedge At(Spare, Trunk)$
- Goal: $At(Spare, Axle)$
- Action Remove:
- Action PutOn: