Planning Representation: Temporal PDDL

Dra. Mª Dolores Rodríguez Moreno

Objectives

Specific Objectives

- Model in PDDL 2.1
- Run SoA planners

Source

- Fox & Long. PDDL2.1: An Extension to PDDL for Expressing Temporal Problems. Journal of AI Research 20 (2003) 61-124
- Eva Onaindia De La Rivaherrera. Planificación Automática. Videos. UPV. https://media.upv.es/

Outline

- **Introduction**
- Temporal Planning
- PDDL 2.X syntax
- Transportation domain
- Airport problem
- Conclusions

Introduction (I)

- Classical planning is restrictive
  - Implicit time assumption
  - Actions no duration
- Need new features for real problems
  - Time
  - Resources
  - Multi-objective
- PDDL is extended: PDDL 2.1

Introduction (II)

- PDDL 2.1 Levels:
  - Level 1: STRIPS version
  - Level 2: the numeric extensions
  - Level 3: the addition of discretised durative actions
  - Level 4: continuous durative actions
  - Level 5: comprised all of the extensions of pddl2.1 and additional components to support the modelling of spontaneous events and physical processes

Introduction (III)

- New
  - Numeric expressions
  - Durative actions
  - Metrics: evaluate the quality of a plan
  - Continuous changes

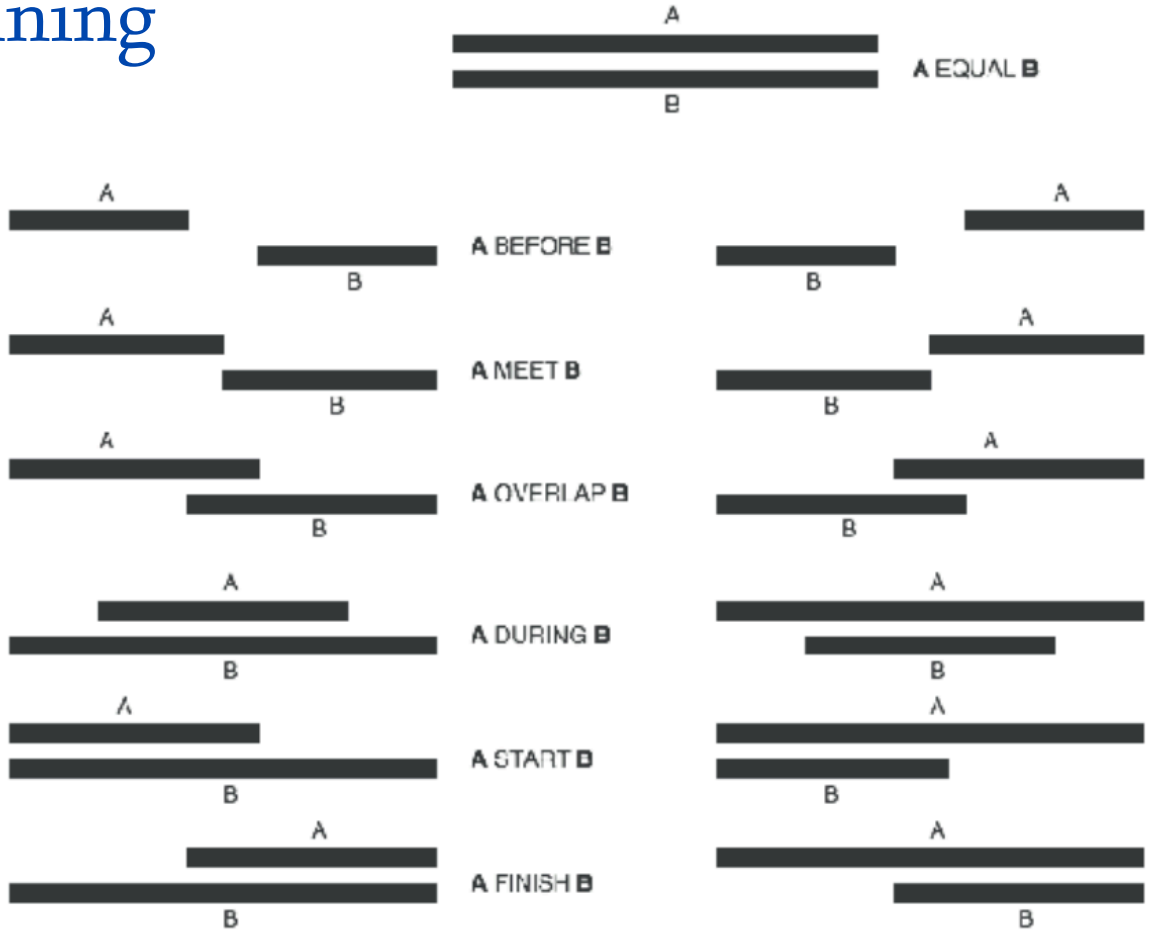
Outline

- Introduction
- **Temporal Planning**
- PDDL 2.X syntax
- Transportation domain
- Airport problem
- Conclusions

Temporal planning

- Sequential planning is not adequate
- What (actions) + When (execution time)
  - Actions synchronization
  - Actions overlapping
  - New optimization criteria
    - Planning steps vs. plan duration (*makespan*)

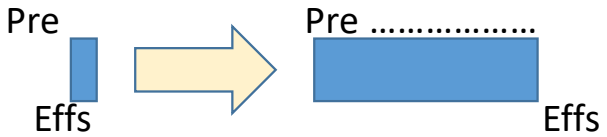
Temporal planning



Source: Flavio S. Corrêa da Silva

Temporal planning: conservative model (I)

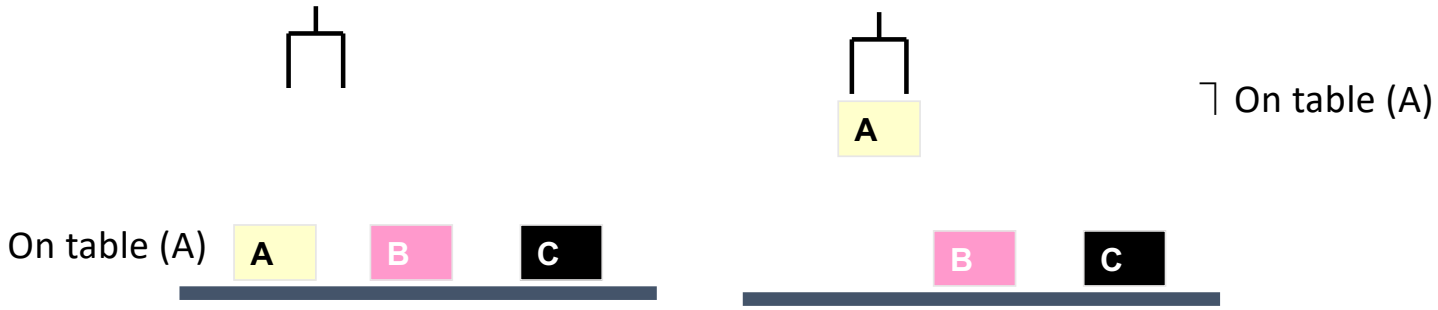
- Easiest way: convert the existing model in a duration model (conservative time model)
- How
  - Preconditions are true at the beginning
  - Effects are true at the end



- Actions are not atomic, allow concurrency (iff no conflicts)
- We cannot know the state of variables in the problem

Temporal planning: conservative model (II)

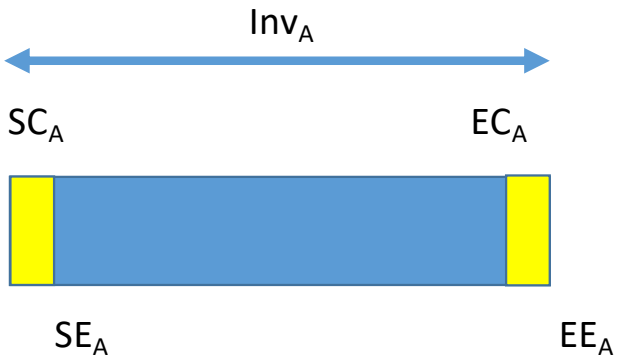
- What happen with pick-up action? Where is A during the process?



- The planner can assume that is all the time on the table (is NOT correct!!!)

Temporal planning: non-conservative model

- Consider when each predicate holds
  - StartCond: (SC_A)
  - $EndCond_A$: (SC_A)
  - $StartEff_A$: (SE_A)
  - $EndEff_A$: (EE_A)
  - $Invariant_A$: (Inv_A)



Outline

- Introduction
- Temporal Planning
- **PDDL 2.X syntax**
- Transportation domain
- Airport problem
- Conclusions

PDDL syntax: Domain

```
(define (domain name)
  (:requirements <require-key> :durative-actions :fluents)
  (:types <typed_list (name)>)
<PDDL list of predicates in the domain>
<PDDL list of functions in the domain>

<PDDL code for first action>
...
<PDDL code for last action>
)
```

PDDL syntax: Actions (I)

(:durative-action <action name>
 :parameters (<list>)
 :duration (= ?duration <number> or (<predicate list>))
 :condition (and (at start/at end/over all (<predicate list>))
 :effect (and (at start/at end/over all (<predicate list>))
)

PDDL syntax: Actions (II)

- To assign
  - assign (not =)
- To add
  - increase
- To subtract
  - decrease

PDDL syntax: Problem

```
(define (problem <problem name>)
  (:domain <domain name>)
  <PDDL code for objects>
  <PDDL code for initial state>
  (= (predicate <parameter list>) number)
  <PDDL code for goal specification>
(:metric minimize (predicate))
)
```

(:metric minimize (total-time))

PDDL syntax: Time in Problem

- Facts in the initial state are true at time 0
- We can specify values ≠ 0

  (at 16 (on a b))
- Recurrent time windows

  (at 10 (f)) (at 20 (not (f))
- Add in requirements → :timed-initial-literals

Outline

- Introduction
- Temporal Planning
- PDDL 2.X syntax
- **Transportation domain**
- Airport problem
- Conclusions

Transportation description

- There are vehicles that need to be moved from different locations
- Vehicles can be accessible or not to move between two different location (cities)
- We know the fuel consumption and the level of the vehicle's tank
- We want to minimize the fuel used by all the vehicles
- Initially, we don't consider durations

Transportation: Domain

```
(define (domain metricVehicle)
 (:requirements :strips :typing :fluents)
 (:types vehicle location)
 (:predicates (at ?v - vehicle ?p - location)
              (accessible ?v - vehicle ?p1 ?p2 - location))
 (:functions  (fuel-level ?v - vehicle)
              (fuel-used ?v - vehicle)
              (fuel-required ?p1 ?p2 - location)
              (total-fuel-used))
 (:action drive
   :parameters (?v - vehicle ?from ?to - location)
   :precondition (and (at ?v ?from)
                      (accessible ?v ?from ?to)
                      (>= (fuel-level ?v) (fuel-required ?from ?to)))
   :effect (and (not (at ?v ?from))
                (at ?v ?to)
                (decrease (fuel-level ?v) (fuel-required ?from ?to))
                (increase (total-fuel-used) (fuel-required ?from ?to))
                (increase (fuel-used ?v) (fuel-required ?from ?to)))
 )
)
```

Transportation: Problem

```
(define (problem mv-example)
 (:domain metricVehicle)
 (:objects
       truck car - vehicle
       Paris Berlin Rome Madrid - location)
 (:init
       (at truck Rome)
       (at car Paris)
       (= (fuel-level truck) 100)
       (= (fuel-level car) 100)
       (accessible car Paris Berlin)
       (accessible car Berlin Rome)
       (accessible car Rome Madrid)
       (accessible truck Rome Paris)
       (accessible truck Rome Berlin)
       (accessible truck Berlin Paris)
       (= (fuel-required Paris Berlin) 40)
       (= (fuel-required Berlin Rome) 30)
       (= (fuel-required Rome Madrid) 50)
       (= (fuel-required Rome Paris) 35)
       (= (fuel-required Rome Berlin) 40)
       (= (fuel-required Berlin Paris) 40)
       (= (total-fuel-used) 0)
       (= (fuel-used car) 0)
       (= (fuel-used truck) 0)
 )
 (:goal (and (at truck Paris)
             (at car Rome))
)
 (:metric minimize (total-fuel-used))
)
```

Transportation

- Time? Something missing?
  - Add a fix duration to the action
  - Add a duration depending on the vehicle

Outline

- Introduction
- Temporal Planning
- PDDL 2.X syntax
- Transportation domain
- **Airport domain**
- Conclusions

Airport domain

- The domain consists of planes and passengers that travel from one city to another
- Model the duration of the actions and the fuel consumption
- Model 4 actions:
  - Board: a person on a plane that is in a city. As a result the person is not in the city and is on the plane
  - Debark: a person from an airplane. As a result the person is in the city, and is not on the plane
  - Fly: from one city to another. The fuel of the plane depends on the distance between the cities and the fuel ratio consumption of the plane. As a precondition, it should be verified

    Fuel >= (distance-between-cities) x (burn-fuel-ratio of the plane)
  - Refuel: an airplane in a city. The duration will depend on the amount of fuel that is left in the tank multiplied by the refuel rate of the plane. As a precondition the level of the fuel in the tank is greater that the actual level, and the plane must be in the city. As an effect, the tank is full.

Outline

- Introduction
- Temporal Planning
- PDDL 2.X syntax
- Transportation domain
- Airport domain
- **Conclusions**

Conclusions

- No model of time in classical planning
- PDDL 2.1 follows non-conservative time model
- What (actions) + When (execution time)
  - Actions synchronization
  - Actions overlapping
  - New optimization criteria
    - Planning steps vs. plan duration (*makespan*)