FF Planner

School of Data and Computer Science
Sun Yat-sen University

Overview

FF Planner

- Fast-Forward, abbreviated FF, is a domain independent planning system developed by Joerg.
- FF can handle classical STRIPS- as well as full scale ADL-planning tasks, to be specified in PDDL.
- FF has competed in the fully automated track of the 2nd International Planning Competition.
- As a result of the competition, FF was granted “Group A distinguished performance Planning System”, and it also won the Schindler Award for the best performing planning system in the Miconic 10 Elevator domain, ADL track.

STRIPS Representation

- STRIPS (Stanford Research Institute Problem Solver) is an automated planner developed by Fikes and Nilsson in 1971.
- The same name was later used to refer to the formal language of the inputs to this planner.
- Actions are modeled as ways of modifying the world.
- An action yields a new KB, describing the new world - the world resulting from executing the action.

STRIPS Actions

- STRIPS represents an action using 3 lists.
    - A list of action preconditions.
    - A list of action add effects.
    - A list of action delete effects.
- These lists contain variables, so that we can represent a whole class of actions with one specification.
- Each ground instantiation of the variables yields a specific action.

STRIPS Action: Example

Example 1

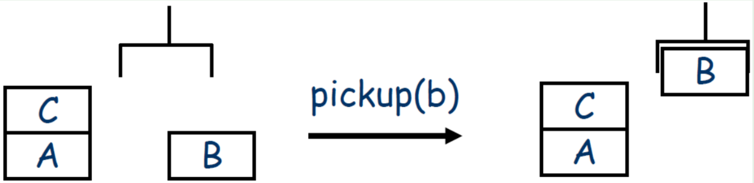
- pickup(X):
  - Pre: handempty, clear(X), ontable(X)
  - Adds: holding(X)
  - Dels: handempty, clear(X), ontable(X)
- "pickup(X)" is called a STRIPS operator.
- A particular instance (e.g."pickup(a)") is called an action.

Operation of a STRIPS action

- For a particular STRIPS action to be applicable to a state
  - Every fact in its precondition list must be true in KB.
  - This amounts to testing membership since we have only atomic facts in the precondition list.
- If the action is applicable, the new state is generated by Removing all facts in Dels from KB, then
- Adding all facts in Adds to KB.

Operation of a Strips Action: Example

Example 2



pre = {handmpty,
clear(b),
ontable(b)}

add = {holding(b)}

del = {handmpty,
clear(b),
ontable(b)}

KB = {handempty
clear(c), clear(b),
on(c,a),
ontable(a), ontable(b)}

KB = { holding(b),
clear(c),
on(c,a),
ontable(a)}

STRIPS Blocks World Operator

- pickup(X) (from the table)
  - Pre: clear(X), ontable(X),handempty
  - Add: holding(X)
  - Dels: clear(X), ontable(X),handempty
- putdown(X) (on the table)
  - Pre: holding(X)
  - Add: clear(X), ontable(X), handempty
  - Del: holding(X)

STRIPS Blocks World Operator (cont'd)

- unstack(X,Y) (pickup from a stack of blocks)
  - Pre: clear(X), on(X,Y), handempty
  - Add: holding(X), clear(Y)
  - Del: clear(X), on(X,Y), handempty
- stack(X,Y) (putdown on a block)
  - Pre: holding(X),clear(Y)
  - Add: on(X,Y), handempty, clear(X)
  - Del: holding(X),clear(Y)

PDDL

The Planning Domain Definition Language (PDDL) supports the following syntactic features:

- Basic STRIPS-style actions
- Conditional effects
- Universal quantification over dynamic universes (i.e., object creation and destruction)
- Domain axioms over stratified theories.
- Specification of safety constraints.
- Specification of hierarchical actions composed of subactions and subgoals.

An Example

Example 3

- This example involving transportation of objects between home and work using a briefcase whose effects involve both universal quantification (all objects are moved) and conditional effects (if they are inside the briefcase when it is moved).
- The domain is described in terms of three action schemata.

```
(define (domain briefcase-world)
  (:requirements :strips :equality :typing :conditional-effects)
  (:types location physob)
  (:constants (B - physob))
  (:predicates (at ?x - physob ?l - location)
               (in ?x ?y - physob))
  ...
```

An Example (cont'd)
domain

- A domain's set of requirements allow a planner to quickly tell if it is likely to be able to handle the domain.
- The briefcase world requires conditional effects, so a straight STRIPS-representation planner would not be able to handle it.
- A keyword (symbol starting with a colon) used in a **:requirements** field is called a requirement flag.
- The domain is said to declare a requirement for that flag.

An Example (cont'd)

domain

- All domains include a few built-in types, such as **object** (any object), and **number**.
- Most domains define further types, such as **location** and **physob** ("physical object") in this domain.
- A constant is a symbol that will have the same meaning in all problems in this domain.
- In this case B - the briefcase - is such a constant. (Although we could have a type briefcase, we don't need it, because there's only one briefcase.)

An Example (cont'd)

action mov-b

```
(:action mov-b
   :parameters (?m ?l - location)
   :precondition (and (at B ?m) (not (= ?m ?l)))
   :effect (and (at b ?l) (not (at B ?m))
                (forall (?z)
                     (when (and (in ?z) (not (= ?z B)))
                           (and (at ?z ?l) (not (at ?z ?m)))))) )
```

- The briefcase can be moved from location ?m to location ?l.
- The preconditions: the briefcase must initially be in the starting location for the action to be legal and it is illegal to try to move the briefcase to the place where it is initially.
- The effect: the briefcase moves to its destination, is no longer where it started, and everything inside the briefcase is likewise moved.

An Example (cont'd)

action put-in

```
(:action put-in
   :parameters (?x - physob ?l - location)
   :precondition (not (= ?x B))
   :effect (when (and (at ?x ?l) (at B ?l))
                 (in ?x)) )
```

- This action definition specifies the effect of putting something (not the briefcase (B) itself!) inside the briefcase.
- If the action is attempted when the object is not at the same place (?l) as the briefcase, then there is no effect.

An Example (cont'd)

action take-out

```
(:action take-out)
   :parameters (?x - physob)
   :precondition (not (= ?x B))
   :effect (not (in ?x)) )
```

The final action provides a way to remove something from the briefcase.

An Example (cont'd)

problem

- This problem supposed that at home one had a dictionary and a briefcase with a paycheck inside it.
- Furthermore, suppose that we wished to have the dictionary and briefcase at work, but wanted to keep the paycheck at home.

```
(define (problem get-paid)
    (:domain briefcase-world)
    (:init (place home) (place office)
           (object p) (object d) (object b)
           (at B home) (at P home) (at D home) (in P))
    (:goal (and (at B office) (at D office) (at P home))))
```

Spare Tire Example

domain-spare-tire.pddl

domain_spare_tire.pddl

```
(define (domain spare_tire)
  (:requirements :strips :equality:typing)
  (:types physob location)
  (:predicates  (Tire ?x - physob)
                (at ?x - physob ?y - location))

(:action Remove
             :parameters (?x - physob ?y - location)
             :precondition (At ?x ?y)
             :effect (and (not (At ?x ?y)) (At ?x Ground)))

  (:action PutOn
             :parameters (?x - physob)
             :precondition (and (Tire ?x) (At ?x Ground)
                                (not (At Flat Axle)))
             :effect (and (not (At ?x Ground)) (At ?x Axle)))
  (:action LeaveOvernight
             :effect (and (not (At Spare Ground)) (not (At Spare Axle))
                     (not (At Spare Trunk)) (not (At Flat Ground))
                     (not (At Flat Axle)) (not (At Flat Trunk)) ))
 )
```

Spare Tire Example

spare-tire.pddl

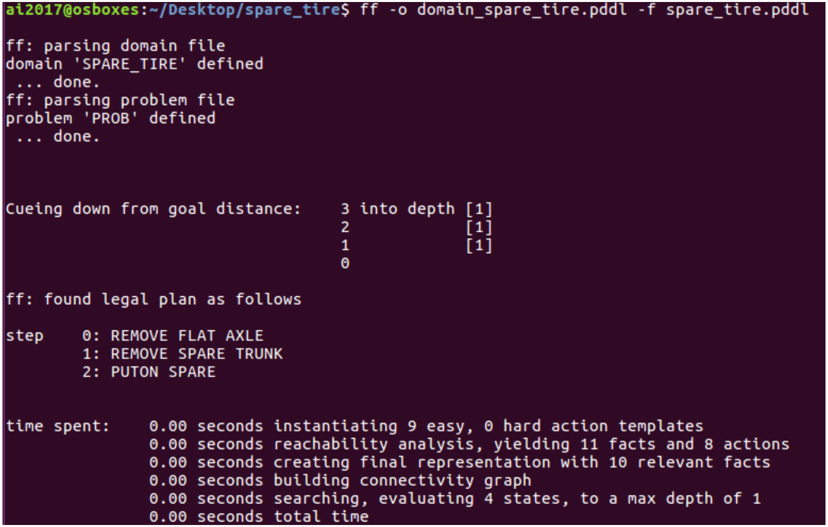
spare_tire.pddl

```
(define (problem prob)
 (:domain spare_tire)
 (:objects Flat Spare -physob Axle Trunk Ground - location)
 (:init (Tire Flat)(Tire Spare)(At Flat Axle)(At Spare Trunk))
 (:goal (At Spare Axle))
)
```

- The goal is to have a good spare tire properly mounted onto the car's axle, where the initial state has a flat tire on the axle and a good spare tire in the trunk.
- 4 actions: removing the spare from the trunk, removing the flat tire from the axle, putting the spare on the axle, and leaving the car unattended overnight.
- Assume the car is parked in a bad neighborhood, so that the effect of leaving it overnight is the tires disappear.

Spare Tire Example

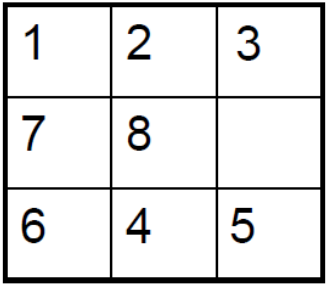
Running Result



```
ai2017@osboxes:~/Desktop/spare_tire$ ff -o domain_spare_tire.pddl -f spare_tire.pddl

ff: parsing domain file
domain 'SPARE_TIRE' defined
 ... done.
ff: parsing problem file
problem 'PROB' defined
 ... done.



Cueing down from goal distance:    3 into depth [1]
                                   2            [1]
                                   1            [1]
                                   0

ff: found legal plan as follows

step    0: REMOVE FLAT AXLE
        1: REMOVE SPARE TRUNK
        2: PUTON SPARE


time spent:    0.00 seconds instantiating 9 easy, 0 hard action templates
               0.00 seconds reachability analysis, yielding 11 facts and 8 actions
               0.00 seconds creating final representation with 10 relevant facts
               0.00 seconds building connectivity graph
               0.00 seconds searching, evaluating 4 states, to a max depth of 1
               0.00 seconds total time
```

Task1

8-puzzle



1	2	3
7	8	
6	4	5

Please complete `domain_puzzle.pddl` and `puzzle.pddl` to solve the 8-puzzle problem.

Task2

Blocks World

There are several blocks. A block can be grasped only when its top is clear. A block can be put on the table or on some other block. We are only allowed to move one block at a time. There are two actions:

- $move(x, y)$: put block x on y. The precondition is that there is no block on the top of x or y.
- $moveToTable(x)$: put block x on the table. The precondition is that block x's top is clear, and it is not on the table.

Please complete `domain_blocks.pddl` and `blocks.pddl` to solve the blocks world problem. You should know the usages of `forall` and `when`.

Reference

- Stuart Russell and Peter Norvig, Artificial Intelligence: A Modern Approach.
- FF homepage, `http://fai.cs.uni-saarland.de/hoffmann/ff.html`
- slides from Sheila McIlraith, CSC384, University of Toronto, Winter 2016

The End