School of Data and Computer Science, Sun Yat-sen University

E07 FF Planner

17341203 Yixin Zhang

October 18, 2019

Contents

1 Examples

1.1 Spare Tire

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.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))
)
```

```
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
```

1.2 Briefcase World

Please refer to `pddl.pdf` at page 2. Please pay More attention to the usages of `forall` and `when`.

For more examples, please refer to `ff-domains.tgz` and `benchmarksV1.1.zip`. For more usages of FF planner, please refer to the documentation `pddl.pdf`.

2 Tasks

2.1 8-puzzle

1	2	3
7	8	
6	4	5

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

domain_puzzle.pddl

```
(define (domain puzzle)
  (:requirements :strips :equality:typing)
  (:types num loc)
  (:predicates ())

(:action slide
            :parameters ()
            :precondition ()
            :effect ()
 )
)
```

domain_puzzle.pddl

```
(define (problem prob)
 (:domain puzzle)
 (:objects )
 (:init )
 (:goal ())
)
```

2.2 Blocks World

现有积木若干，积木可以放在桌子上，也可以放在另一块积木上面。有两种操作：

1. $move(x, y)$：把积木x放到积木y上面。前提是积木x和y上面都没有其他积木。
2. $moveToTable(x)$：把积木x放到桌子上，前提是积木x上面无其他积木，且积木x不在桌子上。

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

domain_blocks.pddl

```
(define (domain blocks)
  (:requirements :strips :typing:equality
                 :universal-preconditions
                 :conditional-effects)
  (:types physob)
  (:predicates
       (ontable ?x - physob)
            (clear ?x - physob)
       (on ?x ?y - physob))

  (:action move
             :parameters (?x ?y - physob)
             :precondition ()
             :effect ()
             )

  (:action moveToTable
             :parameters (?x - physob)
             :precondition ()
             :effect ( )
 )
```

blocks.pddl

```
(define (problem prob)
 (:domain blocks)
 (:objects A B C D E F - physob)
 (:init (clear A)(on A B)(on B C)(ontable C) (ontable D)
  (ontable F)(on E D)(clear E)(clear F)
)
 (:goal (and (clear F) (on F A) (on A C) (ontable C)(clear E) (on E B)
        (on B D) (ontable D)) )
 )
```

Please submit a file named E07_YourNumber.pdf, and send it to ai_201901@foxmail.com

3 Codes and Results

3.1 8-puzzle

3.1.1 Overview

I use nine number objects (num1 – num8 and num0) to indicate the eight numbers and one empty cell, and nine location objects (loc1 – loc8 and loc0) to indicate the nine cells on the puzzle, where locN is numN's target location.

When defining neighbourhood relationship, I just used one-way neighbour, i.e., I defined 'neighbour loc1 loc2' without the existence of 'neighbour loc2 loc1'. This works because I use '(or (neighbour ?from ?to) (neighbour ?to ?from))' to judge neighbourship. This won't be a problem.

3.1.2 Code

domain_puzzle.pddl

```
(define (domain puzzle)
    (:requirements :strips :equality :typing)
    (:types number location)
    (:predicates
        (at ?n - number ?l - location) ; indicates that the number `?n` is at location `?l`
        (neighbour ?l1 ?l2 - location) ; indicates that `?l1` and `?l2` (or `?l2` and
            `?l1`) are neighbours
    )

    (:action slide ; move the number `?num` from the cell `?from` to the cell `?to`
        :parameters (?num - number ?from ?to - location)
        :precondition (and
            (at ?num ?from)
            (at num0 ?to)
            (or (neighbour ?from ?to) (neighbour ?to ?from))
        )
        :effect (and
            (not (at num0 ?to))
            (at num0 ?from)
            (at ?num ?to)
        )
    )
```

```
)
```

problem_puzzle.pddl

```
(define (problem prob)
    (:domain puzzle)

    (:objects
        loc1 loc2 loc3 loc4 loc5 loc6 loc7 loc8 loc0 - location ; locN means numN's target location
        num1 num2 num3 num4 num5 num6 num7 num8 num0 - number ; num0 indicates the empty cell
    )

    (:init
        ; initial configuration
        (at num1 loc1) (at num5 loc2) (at num2 loc3)
        (at num7 loc4) (at num4 loc5) (at num3 loc6)
        (at num8 loc7) (at num0 loc8) (at num6 loc0)

        ; define neighbourhoods (one-way neighbour is enough here)
        (neighbour loc1 loc2) (neighbour loc2 loc3)
        (neighbour loc4 loc5) (neighbour loc5 loc6)
        (neighbour loc7 loc8) (neighbour loc8 loc0)
        (neighbour loc1 loc4) (neighbour loc4 loc7)
        (neighbour loc2 loc5) (neighbour loc5 loc8)
        (neighbour loc3 loc6) (neighbour loc6 loc0)
    )

    (:goal (and
        (at num1 loc1) (at num2 loc2) (at num3 loc3)
        (at num4 loc4) (at num5 loc5) (at num6 loc6)
        (at num7 loc7) (at num8 loc8) (at num0 loc0)
    ))
)
```

3.1.3 Result

It works out correctly. For this simple example, the planner found a solution with eight steps, super cool.

Found Plan (output)

(slide num8 loc7 loc8)

(slide num7 loc4 loc7)

(slide num4 loc5 loc4)

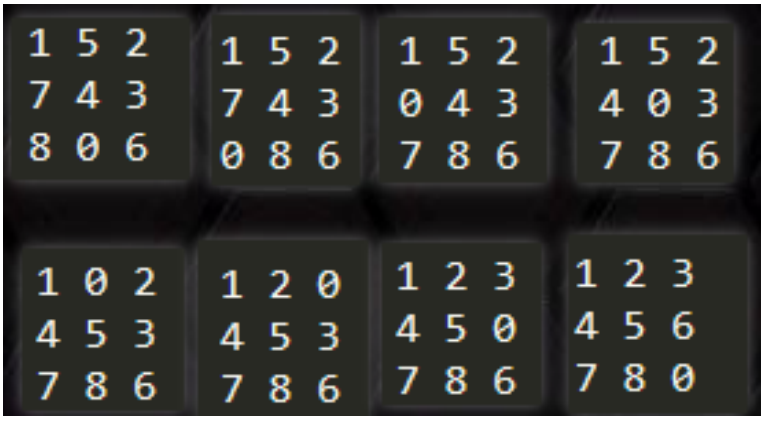
(slide num5 loc2 loc5)

(slide num2 loc3 loc2)

(slide num3 loc6 loc3)

(slide num6 loc0 loc6)

```
(:action slide
  :parameters (num8 loc7 loc8)
  :precondition
    (and
      (at num8 loc7)
      (at num0 loc8)
      (or
        (neighbour loc7 loc8)
        (neighbour loc8 loc7)
      )
    )
  :effect
    (and
      (not
        (at num0 loc8)
      )
      (at num0 loc7)
      (at num8 loc8)
    )
)
```



3.2 Blocks World

3.2.1 Overview

Remember to manage the state of the block under x carefully after moving x away! Here, ‘forall‘ and ‘when‘ statements are used.

3.2.2 Code

domain_blocks.pddl

```
(define (domain blocks)
    (:requirements :strips :typing :equality :universal-preconditions :conditional-effects)
    (:types physob) ; physical object

    (:predicates
        (ontable ?x - physob) ; block ‘?x‘ is on the table
        (clear ?x - physob)  ; there is no block on top of ‘?x‘
        (on ?x ?y - physob)  ; ‘?x‘ is on top of ‘?y‘
    )

    (:action move ; move ‘?x‘ to top of ‘?y‘
        :parameters (?x ?y - physob)
        :precondition (and (clear ?x) (clear ?y) (not (= ?x ?y)))
        :effect (and
            (forall (?z - physob) (when (on ?x ?z) (and (clear ?z) (not (on ?x ?z)))))
            (on ?x ?y) (not (clear ?y))
        )
    )

    (:action moveToTable ; move ‘?x‘ to the table
        :parameters (?x - physob)
        :precondition (and (clear ?x) (not (ontable ?x)))
        :effect (and
            (forall (?z - physob) (when (on ?x ?z) (and (clear ?z) (not (on ?x ?z)))))
            (ontable ?x)
        )
    )
)
```

problem_blocks.pddl

```
(define (problem prob)
    (:domain blocks)

    (:objects A B C D E F - physob)

    (:init
        (clear A) (on A B) (on B C) (ontable C)
        (clear E) (on E D) (ontable D)
        (clear F) (ontable F)
    )

    (:goal (and
        (clear F) (on F A) (on A C) (ontable C)
        (clear E) (on E B) (on B D) (ontable D)
    ))
)
```

3.2.3 Result

Found Plan (output)

- (move f a)
- (movetotable e)
- (move f e)
- (move a f)
- (move b d)
- (move a c)
- (move f a)
- (move e b)

```
(:action move
  :parameters (f a)
  :precondition
    (and
      (clear f)
      (clear a)
      (not
        (= f a)
      )
    )
  :effect
    (and
      (forall (?z - physob)
        (when
          (on f ?z)
          (and
            (clear ?z)
            (not
              (on f ?z)
            )
          )
        )
      )
      (on f a)
      (not
        (clear a)
      )
    )
)
```