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

E06 FF Planner

孙新梦 18340149

October 12, 2020

目录

1 Examples

1.1 Spare Tire

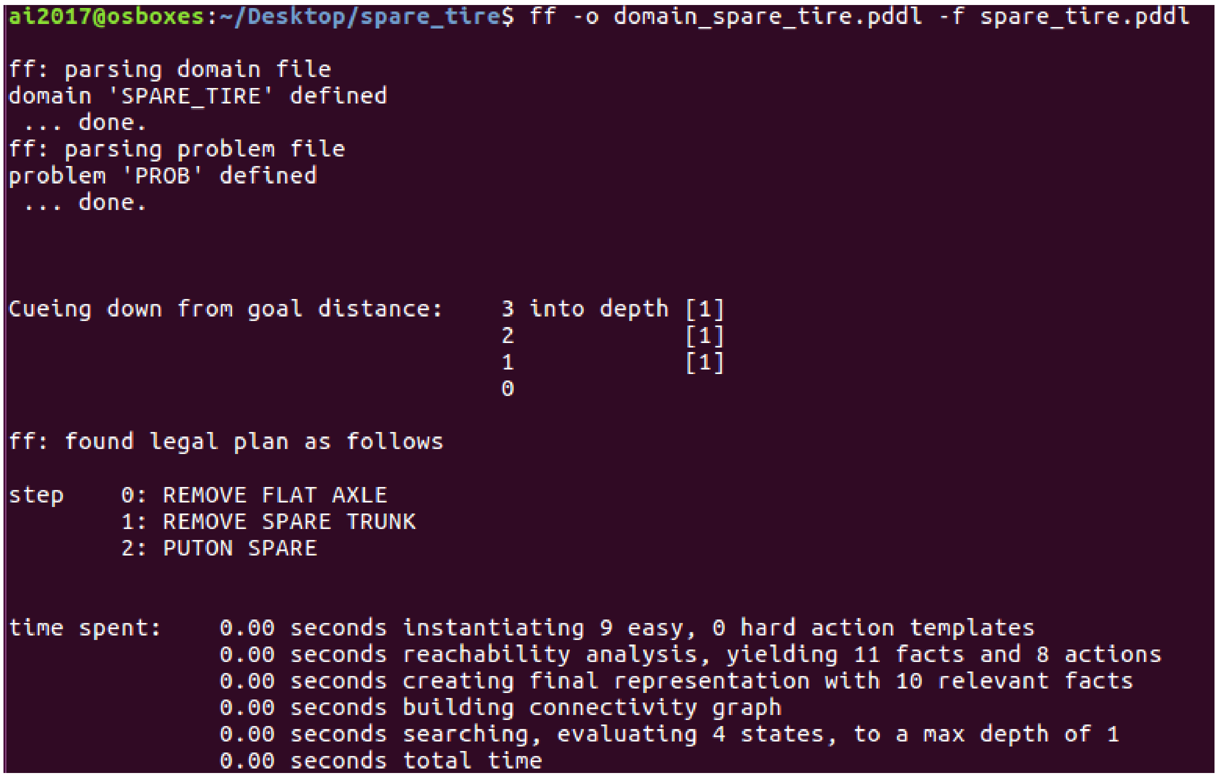
domain_spare_tire.pddl

```
(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

Planning in the blocks world is a traditional planning exercise, and you can recall what we have introduced in the theory course.
There are a collection of blocks: a block can be on the table, or on the top of another block.
There are three predicates:

- *clear(x)*: there is no block on top of block x;
- *on(x,y)*: block x is on the top of block y;
- *onTable(x)*: block x is on the table

There are two actions in this task:

- *move(x,y)*: move block x onto block y, provided that both x and y are clear;
- *moveToTable(x)*: move block x on to the table, provided that x is clear and x is not on the table;

Give initial state and goal state, find the actions change the initial state to the goal state.

In this task, 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 E06_YourNumber.pdf, and send it to ai_2020@foxmail.com

3 Codes and Results

3.1 8 Puzzle

domain_puzzle.pddl

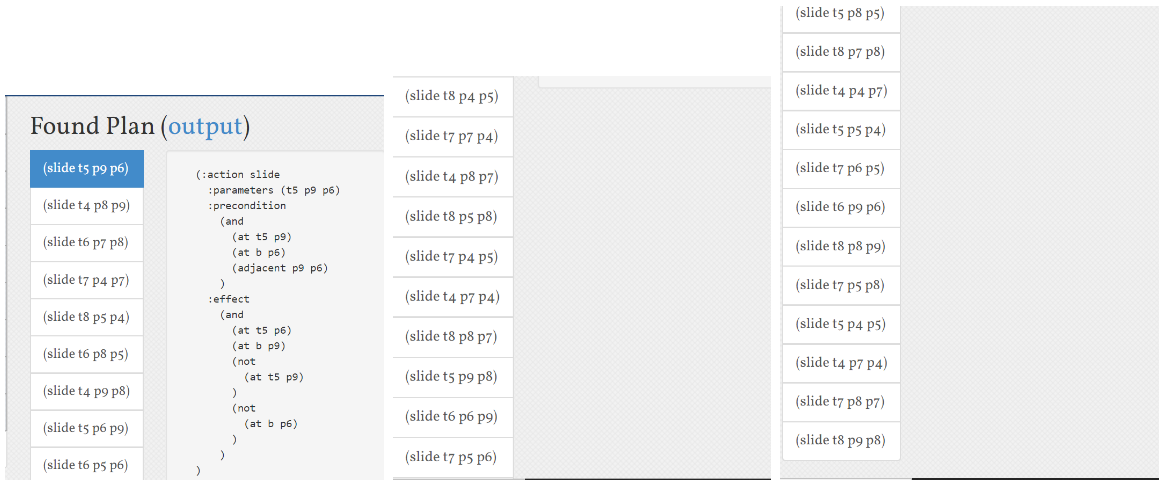
```
(define (domain puzzle)


      (:requirements
           :strips
           :equality
```

```
        :typing
    )

    (:types
        num loc
    )
    (:constants b - num)
    (:predicates
        (adjacent ?x ?y - loc)
        (at ?x - num ?y - loc)
    )

    (:action slide
        :parameters (?t  - num ?x ?y - loc)
        :precondition (and (at ?t ?x ) (at b ?y) (adjacent ?x ?y))
        :effect (and (at ?t ?y) (at b ?x) (not(at ?t ?x)) (not(at
           b ?y)) )
    )
)
```

problem_puzzle.pddl

```
(define (problem prob)
    (:domain puzzle)
    (:objects
        t1 t2 t3 t4 t5 t6 t7 t8 - num
        p1 p2 p3 p4 p5 p6 p7 p8 p9 - loc
    )

    (:init
        (at t1 p1)
        (at t2 p2)
        (at t3 p3)
        (at t7 p4)
        (at t8 p5)
        (at b p6)
        (at t6 p7)
        (at t4 p8)
        (at t5 p9)

        (adjacent p1 p2)
        (adjacent p1 p4)
        (adjacent p2 p1)
        (adjacent p2 p3)
        (adjacent p2 p5)
        (adjacent p4 p1)
        (adjacent p4 p5)
        (adjacent p4 p7)
        (adjacent p5 p2)
        (adjacent p5 p4)
        (adjacent p5 p6)
        (adjacent p5 p8)
        (adjacent p6 p3)
        (adjacent p6 p5)
        (adjacent p6 p9)
        (adjacent p7 p4)
        (adjacent p7 p8)
        (adjacent p8 p7)
        (adjacent p8 p5)
        (adjacent p8 p9)
        (adjacent p9 p6)
        (adjacent p9 p8)
    )

    (:goal(and (at t1 p1)
               (at t2 p2)
               (at t3 p3)
```

```
                        (at t4 p4)
                        (at t5 p5)
                        (at t6 p6)
                        (at t7 p7)
                        (at t8 p8)
                        (at b p9)
                )
        )
)
```



Found Plan (output)

```
(slide t5 p9 p6)
(slide t4 p8 p9)
(slide t6 p7 p8)
(slide t7 p4 p7)
(slide t8 p5 p4)
(slide t6 p8 p5)
(slide t4 p9 p8)
(slide t5 p6 p9)
(slide t6 p5 p6)
```

```
(:action slide
  :parameters (t5 p9 p6)
  :precondition
    (and
      (at t5 p9)
      (at b p6)
      (adjacent p9 p6)
    )
  :effect
    (and
      (at t5 p6)
      (at b p9)
      (not
        (at t5 p9)
      )
      (not
        (at b p6)
      )
    )
)
```

```
(slide t8 p4 p5)
(slide t7 p7 p4)
(slide t4 p8 p7)
(slide t8 p5 p8)
(slide t7 p4 p5)
(slide t4 p7 p4)
(slide t8 p8 p7)
(slide t5 p9 p8)
(slide t6 p6 p9)
(slide t7 p5 p6)
```

```
(slide t5 p8 p5)
(slide t8 p7 p8)
(slide t4 p4 p7)
(slide t5 p5 p4)
(slide t7 p6 p5)
(slide t6 p9 p6)
(slide t8 p8 p9)
(slide t7 p5 p8)
(slide t5 p4 p5)
(slide t4 p7 p4)
(slide t7 p8 p7)
(slide t8 p9 p8)
```

3.2 Blocks

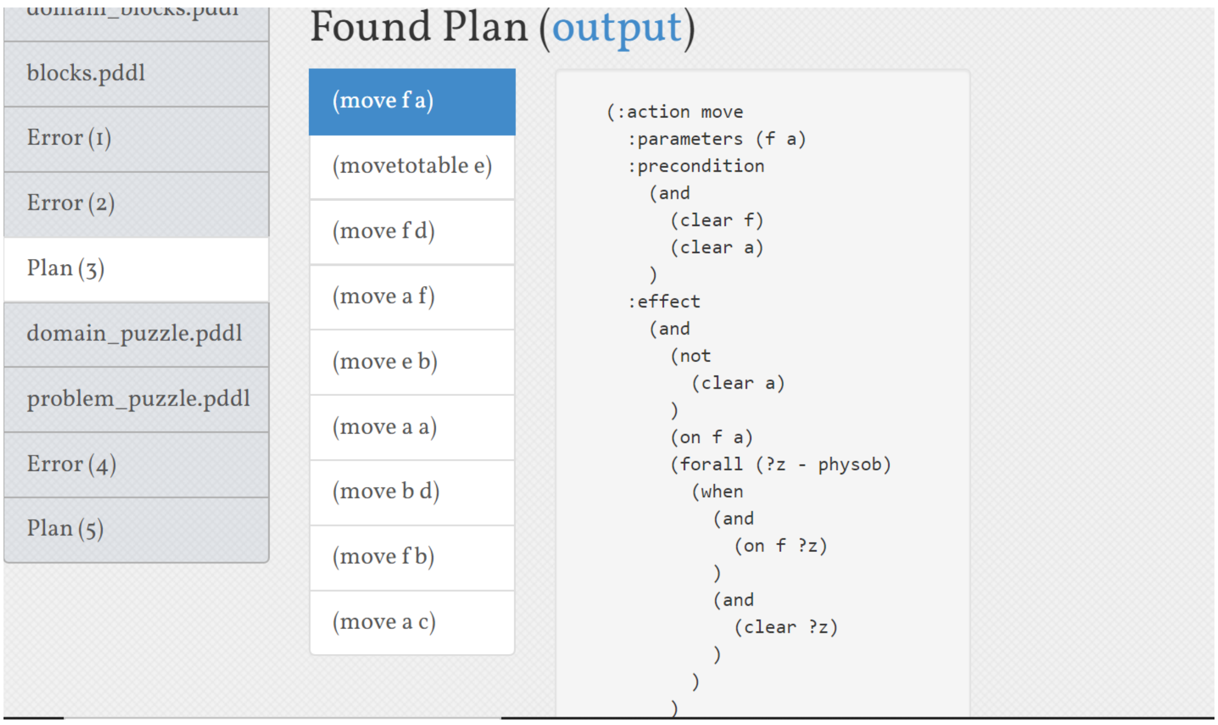
domain_blocks.pddl

```
;Header and description

(define (domain blocks)

;remove requirements that are not needed
(:requirements :strips :typing :negative-preconditions :
    conditional-effects :universal-preconditions :equality)

(:types ;todo: enumerate types and their hierarchy here, e.g. car
    truck bus - vehicle
    physob
)

(:predicates ;todo: define predicates here
    (ontable ?x - physob)
    (clear ?x - physob)
    (on ?x ?y - physob)
)

;define actions here
(:action move
    :parameters (?x ?y - physob)
    :precondition (and (clear ?x) (clear ?y))
    :effect (and (not(clear ?y) ) (on ?x ?y)
                (forall (?z - physob)
                    (when (and (on ?x ?z) )
                        (and (clear ?z))
                    )
                )
            )
)

(:action moveToTable
    :parameters (?x - physob)
    :precondition (and (clear ?x) (not (ontable ?x)))
    :effect (and (ontable ?x)
                (forall (?z - physob)
                    (when (and (on ?x ?z) )
                        (and (clear ?z))
                    )
                )
            )
)
```

```



)
```

problem_blocks.pddl

```
(define (problem prob)
    (:domain blocks)
    (:objects A B C D E F - physob
    )

    (:init
    ;todo: put the initial state's facts and numeric values here
        (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)
        )
        )

)
```



domain_blocks.pddl
blocks.pddl
Error (1)
Error (2)
Plan (3)
domain_puzzle.pddl
problem_puzzle.pddl
Error (4)
Plan (5)

Found Plan (output)

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

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

4 My thoughts

这次实验感觉比较顺利，也很快就完成了。最近非常喜欢上AI实验课，因为每次都学的是那种不是很难但是有点挑战度的实用方法与技术，涨了很多见识。

规划指的是我们的agent能够有目的的行动并且知道自己动作会给世界带来怎样的变化，首先学习了情景演算的方法，用归结的方法来规划动作序列，但是问题在于一个动作改变的事实比起不变的事实不过是冰山一角九牛一毛，所以会导致框架问题。

之后学习了STRIPS，把前提满足时的动作的改变效果加入知识库——从F变成T的加入，从T变成F的删掉，这样可以通过是否在知识库中很简便验证一个命题的正确与否。

但是STRIPS没有条件效果，也即是不能表示一些当x为...的时候，为世界执行怎样变化。由此引入ADL语言

这回使用的PDDL就是一个自动规划器，当我们用ADL风格语言提供domian知识，定义使用的谓词，动作，变量类型，再提供我们的问题，定义变量，提供初始状态，目标状态之后，自动规划器为我们找到动作序列。

需要注意的是PDDL的编写需要注意很多小细节，比如-与变量类型之间是有一个空格，再比如谓词名后面不能加括号来表示，这些都很繁杂，但是比起他带来的便捷来说，也值得。我们需要继续提高用逻辑表示现实世界的能力，对前提与效果要想的更多更深入。