Artificial Intelligence: Homework on Planning

Edoardo Ghini

December 20, 2016

Dipartimento di Ingegneria dell'Università di Roma La Sapienza

Contents

I Introduction 1

II Development 2

1 Vanilla Grid 2
1.1 Strips formulation . . . 2
1.1.1 Predicates . . . 2
1.1.2 Actions . . . 2
1.1.3 Initial and goal states . . . 3
1.2 Planner result . . . 3

2 Extended Grid 4
2.1 Strips formulation . . . 4
2.1.1 Predicates . . . 4
2.1.2 Actions . . . 4
2.1.3 Initial and goal states . . . 7
2.2 Planner result . . . 8

3 POP 10
3.1 Design choices . . . 10
3.2 Abstraction level . . . 10
3.3 Conflict resolution . . . 10
3.4 Graph representation . . . 10
3.5 Plan order . . . 12

4 HTN 13
4.1 Primitives . . . 13
4.2 Hierarchical graph . . . 13

Part I

Introduction

? ¬ The most important factor that deserves to be taken into consideration approaching to a logic based problem is, certainly, the capability to represent the problem through a coherent domain abstraction.

This essential concept is at base of which I am going to debate.
In fact, there will be the application of the planning techniques like fast-forward planning, partial ordering and hierarchical planning, on an abstract model built onto a real world problem.

Part II

Development

1 Vanilla Grid

At first the model has been built according to a simplification of the problem : a robot has to navigate through a gris world from a start position to the goal position. It would have to find the path toward the solution avoiding possible walls.

1.1 Strips formulation

In order to obtain a representation which could bring to a resolution of the problem I used the formalism of STRIPS.

1.1.1 Predicates

```
        connected( square-1  square-2)
        white ( square)
        black ( square)
        at ( who where)
        place ( where )
```

Code 1: vanilla grid predicates

1.1.2 Actions

Accordingly with the extreme minimality of the domain, only one action has been enough to be able to create a successfull plan

```
Action: Move( who, from, to )
        Precondition:
                At( who, from ),
                  place ( from),
                  place ( to),
                  connected ( from, to),
                  white( to)
         Effect:
                  at ( who to),
                ¬ at (who from)

```

Code 2: vanilla grid actions

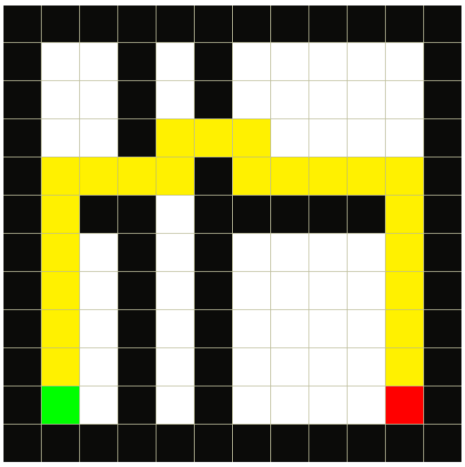

Figure 1

1.1.3 Initial and goal states

```
Initial State:
        place( square0-0) ? place( square0-1) ? place( square1-0) ?
            place( square1-1) ? ...
...     connected( square0-0 square 0-1) ? connected( square0-0
    square1-0)  ? ...
...     black( square0-0) ? black(square 0-1) ? black( square1-0) ?
    white( square1-1) ? ...
...     at( robot square1-1)
Goal State:
        at( robot square10-1)
```

Code 3: vanilla goal and initial state

About the initial configuration, I defined every single square of the grid and also each link of reachability between each other and also the fact that there where white squares that correspond to the floor of the house and, similarly black cells that stand for walls.

1.2 Planner result

In fig(1), it cold be seen the path found from the planner in order to satisfy the goal precondition.

2 Extended Grid

For the purpose of obtaining a more complex domain, the real world has changed and other objects and properties have been added.

2.1 Strips formulation

In the same fashion as the vanilla problem, I designed a representation of the real world through predicates and actions.

2.1.1 Predicates

Unlike in PDDL, in STRIPS we can't explicitly express equality constraints and for this reason I have had to specify objects like seat and tray with unambiguous predicates.

In other words, I added redundancy to represent multiple similar objects in the model.

```
        carry ( who, what),
        seat1 ( seat),
        seat2 ( seat),
        table ( where),
        robot ( who),
        white ( where),
        black ( where),
        closet ( where),
        dishwasher ( where),
        finished ( what),
        washingmachine ( where),
        not-started-yet ( what),
        clean ( what),
        at ( who, where),
        tray1 ( what),
        tray2 ( what),
        cloth ( what),
        conn ( square-1 square-2),
        unloaded ( who),
        clear ( where)
```

Code 4: extended grid predicates

2.1.2 Actions

```

Action: Move( who, from, to)
         Precondition:
                robot ( who),
                 white ( from),
                 white ( to),
                 at ( who, from),
                 conn (from, to)
        Effect:
                ¬ at ( who, from),
                 at ( who, to)

Action: Put (who, what, where, near)
    Preconditions:
                 carry ( who, what),
                 white ( near),
                 at ( who, near),
                 conn ( near where)
    Effect:
                unloaded ( who),
                 ¬ carry ( who, what),
                 at ( what, where)
```

Code 5: exetended grid actions

Similarly as what I have done with run actions, I chose to divide the action take in four sub-actions and also I met the same problem as before: there was need of negative preconditions.

In order to overcame this issue I have created another dual predicate, but this time the oldest (loaded(who)) would have been substituted completely by its logic negation : unloaded(who).

```
Action: Take-cloth-from-table ( who, what, where, near, seat-1, seat
    -2)
        Precondition:
                  robot( who),
                   cloth ( what),
                   table ( where),
                   white ( near),
                   seat1( seat-1),
                   seat2 ( seat-2),
                   clear ( seat-1),
                   clear ( seat-2),
                   at ( who, near),
                   conn ( near, where),
                   at ( what, where),
                   unloaded ( who)
           Effect:
                  carry ( who, what),
                   ¬ unloaded ( who),
                   ¬ at( what where),
                   clear ( where)

Action:  Take-tray-from-seat ( who, what, where, near )
           Precondition:
                  robot ( who),
                   tray ( what),
                   seat (where),
                   white ( near),
                   conn ( near, where),
                   at ( who, near),
                   at ( what, where),
                  unloaded ( who)
             Effect:
                 ¬ unloaded ( who),
                  carry ( who, what),
                  ¬ at ( what, where ),
                  clear ( where)

Action:  Take-cloth-from-waschingmachine ( who, what, where, near)
        Precondition:
                robot ( who),
                 unloaded ( who),
                 cloth ( what),
                 washingmachine ( where),
                 white ( near),
                 conn ( near, where),
                 at ( who, near),
                 at ( what, where),
                 finished ( where)
        Effect:
                 ¬ unloaded ( who),
                  carry ( who, what),
                  ¬ at ( what, where)

Action:   Take-tray-from-dishwasher  (who, what, where, near)
        Precondition:
                robot( who),
                 unloaded( who),
                 tray ( what),
                 dishwasher ( where),
                 white ( near),
                 conn ( near, where),
                 at ( who, near),
                 at ( what, where),
                 finished ( where)
        Effect:
                ¬ unloaded ( who),
                carry ( who, what),
                 ¬ at ( what, where)

```

Code 6: vanilla grid take actions

At a certain point, I thought that there would be some reasons to support the division of the run action in two sub-actions, allowing an easier management of the preconditions.

In addition, during the definition of both the actions I needed to solve another problem: STRIPS language doesn't allow to define actions with negative preconditions.
Hence I introduced a dual predicate in order to use it in the preconditions in the place of the negated one. This was the case with not-started-yet(what) and finished (what).

```

Action:  Run-washingmachine ( who, what, near, inside)
        Preconditions:
                robot ( who),
                 washingmachine ( what),
                 white ( near),
                 conn ( near, what),
                 at ( who, near),
                 not-started-yet ( what),
                 cloth ( inside),
                 at ( inside, what )
         Effect:
                finished ( what),
                 clean ( inside),
                 ¬ not-started-yet( what)
Action:   Run-dishwasher  ( who, what, near, inside1, inside2)
        Precondition:
                robot ( who),
                 dishwasher ( what),
                 white ( near),
                 conn ( near, what),
                 at ( who, near),
                 not-started-yet ( what),
                 tray1 ( inside1),
                 at ( inside1, what),
                 tray2 ( inside2),
                 at ( inside2, what)
        Effect:
                finished ( what),
                 clean ( inside1),
                 clean ( inside2),
                  ¬ not-started-yet( what)


```

Code 7: extended grid run actions

2.1.3 Initial and goal states

Likewise I have done in the simpler vanilla domain, I initialised the problem defining grid squares and their connections, objects positions and some predicates which where true from the beginning.

Differently from vanilla world, there isn't the predicate ' place (where) ' anymore because there where more specific places to model in the domain, for instance a table, two seats, a closet and so on and so forth.

In fig(2), there is a graphical representation of the model.

```
        Initial State:
                connected( square0-0 square 0-1) ^ connected( square
                    0-0 square1-0)  ? ...
        ...     black( square0-0) ? black(square 0-1) ? black(
            square1-0) ? white( square1-1) ? ...
```

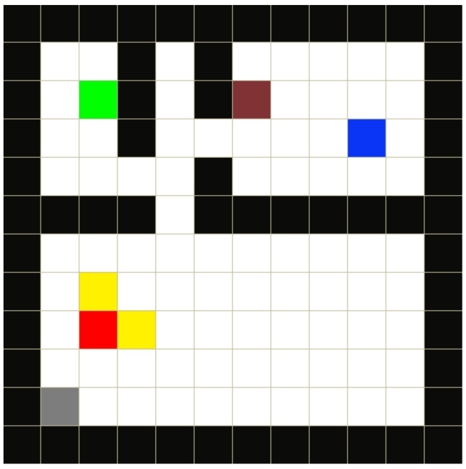

Figure 2

```
        ...     at( robot1 square7-1) ? seat1( square3-3) ? seat2(
            square2-4) ? table( square2-3) ?
                ? dishwasher( square9-8) ? washingmachine( square2-9
                    ) ? closet( square6-9) ?
                ? tray1( tray-1) ? at( tray-1 square3-3) ? tray2(
                    tray-2 square2-4) ? cloth( cloth1) ?
                ? at( cloth1 node2-3) ? robot( robot1) ?
                ? not-started-yet( square2-9) ? not-started-yet(
                    square9-8)

        Goal State:
                at( tray-1 square6-7) ? at( tray-2 square6-9) ? at(
                    cloth1 square2-3) ?
                ? clean( tray-1) ? clean( tray-2) ? clean( tray-3)
```

Code 8: vanilla goal and initial state

2.2 Planner result

When the resolver tried to find a plan for this domain modelled in PDDL, he succeed without instantiating hard actions.
Under those circumstances the search space will be smaller, hard actions should be avoided, consequently it is advisable to not introduce ”or” clauses in the PDDL implementation.

```

        0.77 seconds instantiating 6921932 easy, 0 hard action
            templates
              1.91 seconds reachability analysis, yielding 795
                  facts and 1216 actions
              0.00 seconds creating final representation with 502
                  relevant facts, 0 relevant fluents
```

```
                0.00 seconds computing LNF
                0.01 seconds building connectivity graph
                0.06 seconds searching, evaluating 744 states, to a
                    max depth of 20
                2.75 seconds total time
```

Code 9: plan generated on extended grid

3 POP

Subsequently I simulated a partial order planning on the extended grid domain.

3.1 Design choices

I tried to be coherent with the STRIPS and PDDL formulation, in fact the graph can be seen as as a composition of three main threads which corresponds at three similar action sequences regarding the three object that are involved: two trays and one cloth.

3.2 Abstraction level

One significative choice that i have made in order to simplify the domain was to consider the action move without preconditions. This, in fact, means that it could be executed at any time from any situations.

3.3 Conflict resolution

For this reason, an action move without precondition, the plan computation turned to be smooth because I did't meet conflicts of some sorts.
But, at the same time, due to the fact that the robot could move in any situation toward anywhere, there are a lot of clubbering links between actions like put and take or also run and the single action move toward a different location.
For instance I represented only one case of clubber presence in fig(3) with green arrows: the robot could decide to pursue another object in order to clean the house like the first tray or the closet if the first tray has already been taken instead of going on taking the just cleaned second tray.

3.4 Graph representation

In the afore mentioned fig(3) there is the graphical representation

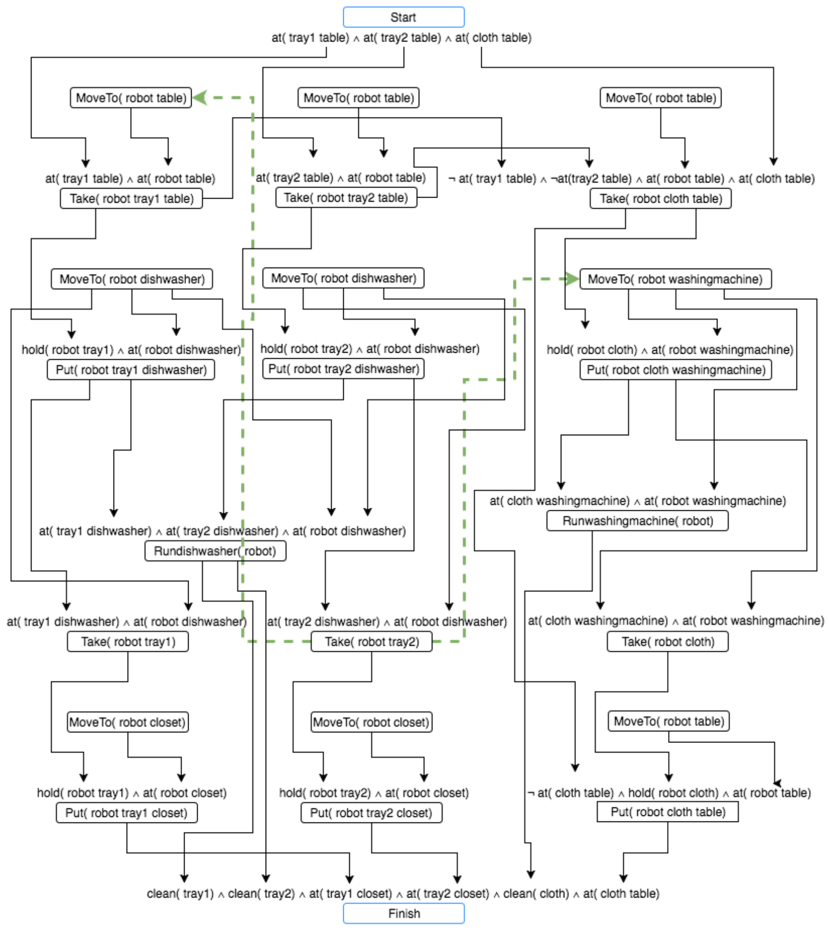
Start
at(tray1 table) ∧ at(tray2 table) ∧ at(cloth table)
MoveTo(robot table)
MoveTo(robot table)
MoveTo(robot table)
at(tray1 table) ∧ at(robot table)
Take(robot tray1 table)
at(tray2 table) ∧ at(robot table)
Take(robot tray2 table)
¬ at(tray1 table) ∧ ¬at(tray2 table) ∧ at(robot table) ∧ at(cloth table)
Take(robot cloth table)
MoveTo(robot dishwasher)
MoveTo(robot dishwasher)
MoveTo(robot washingmachine)
hold(robot tray1) ∧ at(robot dishwasher)
Put(robot tray1 dishwasher)
hold(robot tray2) ∧ at(robot dishwasher)
Put(robot tray2 dishwasher)
hold(robot cloth) ∧ at(robot washingmachine)
Put(robot cloth washingmachine)
at(cloth washingmachine) ∧ at(robot washingmachine)
Runwashingmachine(robot)
at(tray1 dishwasher) ∧ at(tray2 dishwasher) ∧ at(robot dishwasher)
Rundishwasher(robot)
at(tray1 dishwasher) ∧ at(robot dishwasher)
Take(robot tray1)
at(tray2 dishwasher) ∧ at(robot dishwasher)
Take(robot tray2)
at(cloth washingmachine) ∧ at(robot washingmachine)
Take(robot cloth)
MoveTo(robot closet)
MoveTo(robot closet)
MoveTo(robot table)
hold(robot tray1) ∧ at(robot closet)
Put(robot tray1 closet)
hold(robot tray2) ∧ at(robot closet)
Put(robot tray2 closet)
¬ at(cloth table) ∧ hold(robot cloth) ∧ at(robot table)
Put(robot cloth table)
clean(tray1) ∧ clean(tray2) ∧ at(tray1 closet) ∧ at(tray2 closet) ∧ clean(cloth) ∧ at(cloth table)
Finish

Figure 3

3.5 Plan order

It will follow the sequence of actions which I have computed during planning operations: Order MoveTo(robot table), Take(robot tray1), MoveTo(robot dishwasher), Put(robot tray1 dishwasher), MoveTo(robot table), Take (robot tray2), MoveTo(robot dishwasher), Put (robot tray2 dishwasher), Rundishwasher(robot), MoveTo(robot table), Take(robot cloth), MoveTo(robot washingmachine), Put(robot cloth washingmachine), Runwashingmachine(robot), MoveTo(robot dishwasher), Take(robot Tray1), MoveTo(robot closet), Put (robot tray1 closet), MoveTo(robot dishwasher) ,Take (robot tray2), MoveTo(robot closet), Put(robot tray2 closet), MoveTo(robot washingmachine), Take(robot cloth), MoveTo(robot table), Put (robot cloth table)

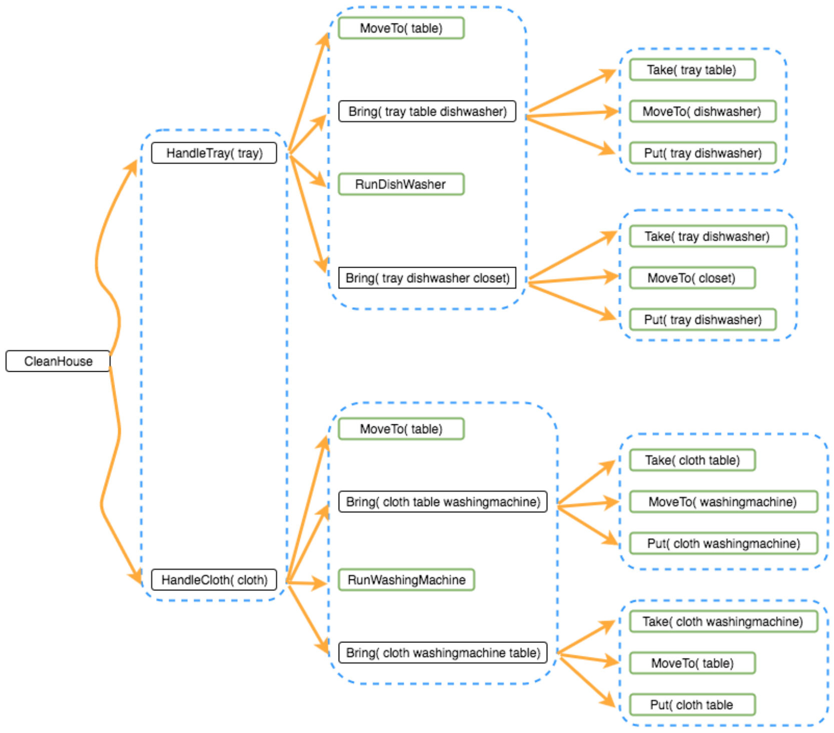

Figure 4

4 HTN

4.1 Primitives

I maintained the actions defined in POP and STRIPS to model HTN primitive actions, on the other hand I tried to relax the representation removing some parameters from them.

4.2 Hierarchical graph

The fig(4) shows how I designed a higher abstraction of the extended grid domain.