Report

Alice Nardelli, Iacopo Pietrasanta, Giovanni Di Marco.

Motivations for the design of our model

The model created is the result of our works. We did not inspired to any already developed work.

We firstly evaluate to use PDDL+ for modelling our domain. Secondly we reasoned that everything in our model needs to be executed because is chosen by the planner and not because some preconditions become true. According to that the use of durative_actions instead of processes would be more appropriate. This made us choose PDDL2.1.

At the purpose of modelling our domain with PDDL2.1 we read some papers. The one shown below is one example.

Fox, Maria, and Derek Long. "PDDL2. 1: An extension to PDDL for expressing temporal planning domains." *Journal of artificial intelligence research* 20 (2003): 61-124.

Results

The line to launch the domain and the problem is:

```
./lpg++ -o domain.pddl -f problem.pddl -n 3 -force_neighbour_insertion -seed number_of_seed
```

The only working option is -seed. This allows to stabilize the results. For all problems we evaluated different values of seed we have chosen the value 1000000 for the best and most stable results.

Problem 1

```
./lpg++ -o Domain.pddl -f Problem1.pddl -n 3 -force_neighbour_insertion  -seed 1000000
```

```
Number of actions: 110
Number of conditional actions:  0
Number of facts: 25

Third solution found after 150 search step
Total time:      0.14
Search time:     0.14
Actions:         18
Duration:        16.000
Plan quality:    18.000
Total Num Flips: 607
```

Problem2

```
./lpg++ -o Domain.pddl -f Problem2.pddl -n 3 -force_neighbour_insertion  -seed 1000000
```

```
Number of actions:     124
Number of conditional actions :       0
Number of facts:      31

Third solution found after 100 search step
```

```
Total time:      0.49
Search time:     0.49
Actions:         25
Duration:        29.500
Plan quality:    25.000
Total Num Flips: 1450
```

Problem3

```
./lpg++ -o Domain.pddl -f Problem3.pddl -n 3 -force_neighbour_insertion  -seed 1000000
```

```
Number of actions :     124
Number of conditional actions :      0
Number of facts:      31

Third solution found after 150 search step
Total time:      0.11
Search time:     0.11
Actions:         27
Duration:        21.000
Plan quality:    27.000
Total Num Flips: 357
```

Problem4

```
./lpg++ -o Domain.pddl -f Problem4.pddl -n 3 -force_neighbour_insertion  -seed 1000000
```

```
Number of actions:     152
Number of conditional actions:      0
Number of facts:      43

Third solution found after 150 search step
Total time:      3.79
Search time:     3.79
Actions:         53
Duration:        55.500
Plan quality:    53.000
Total Num Flips: 12649
```

Domain description

For what concerns types, we have *drink*, *agent*, e *table* (which also describes the bar counter). Note that the barman is not modeled as an *agent* but we decided to only model its actions.
For what concerns predicates, *not-occupied* state if a table is already occupied by a waiter (*not-occupied* has been introduced for multi-agent extension n^2). *bar-counter* indicates which table is the bar counter. ***free*** indicates if the barman can make a drink or is already working. ***carrying*** keeps track of which drinks are actually carried by a waiter. ***hot*** and ***cold*** initially were simply ***hot ?d - drink*** but since we adopted extension n^2 we needed to have a single waiter serving a table: since we know the orders from the beginning, the easiest way to achieve this was to simply modify these predicates in ***hot ?d - drink ?a - agent***, basically fixing the agent that will have to handle a particular drink.

For what regards functions, most are self explanatory, while ***on-counter*** keeps track of the number of drink at the bar counter (ready to be picked and brought to tables), ***tot-cold*** and ***tot-hot*** are counters for the barman, they express the total number of hot and cold drink that needs to be prepared, they usually start at a certain value and are decremented once the barman completes drinks. ***carry-agent*** keeps track of the number of carried drinks of an agent, while ***carry-capacity*** it's the maximum agent carry capacity at that moment, and can be altered by picking/putting the tray. ***ordered-hot*** and ***ordered-cold*** are information about drink ordered by a specific table, they are decremented each time a drink is delivered to the table that ordered it. ***surf_m*** indicates the dirty surface of a table : we adopted this design because with only this function , we can handle the problem of cleaning a table and also of specifying the right duration of the cleaning durative-action.
For the extension number 4 we noticed that in order to introduce biscuits we didn't need to develop a new type (as for drinks) we all the consequent needed predicates. We decided to introduce functions to model this extension. ***needed-biscuits ?t - table ?a - agent*** is a counter to work very similar to ***ordered-hot*** and ***ordered-cold,*** also assuring that the constraint of uniqueness between agent and table is respected. ***biscuitable ?t – table*** is increased every time a cold drink is put on the table. ***carry-biscuits ?a – agent*** state that an agent can carry multiple biscuits and as long as *carry-biscuits+carry-agent<agent_capacity* can carry both biscuits and drinks.

Simple action in the model are pretty straight forward and easy to understand.
For what regards durative actions
in the ***move*** the time of effects is significative:

(at end(at-agent ?a ?t2))
(at start(not (at-agent ?a ?t1) //this prevents agent performing different ***move*** simultaneously
(at start(not-occupied ?t1))
(at start(not(not-occupied ?t2) //prevent different agents trying to ***move*** to same table

the ***make-cold-drink*** and ***make-hot-drink*** are straight forward.
In the ***cleaning*** action, it is made sure in the conditions that an agent can clean a table only after all the other necessary operations have already been performed (all drinks need to be delivered and consumed ad biscuits served). As said the use of ***surf_m*** allowed us to have a single cleaning action also for table with different dimensions.
The two ***put-down-consume*** actions are pretty significant : initially they were simple immediate actions for putting down drinks , but when we adopted extension n^3 we had to deal with the fact that delivered drinks are consumed in 4 time instants, our solution was to incorporate this in the put down action transforming it in a durative action. This also solved the fact that drinks <u>must</u> be consumed , and concept of obligation is not always easy to achieve with actions. After a cold drink is delivered , the waiter will also deliver a biscuit to that table, and for that we decided to develop both atomic actions for picking and delivering the biscuit (with a need of a ***move*** from the bar to the ***biscuitable*** table) and a big macro durative action that includes all the atomic actions. These allows us to have a great flexibility, giving the possibility to the planner of choosing the macro when it's convenient, or to still use atomic actions that grants him much more flexibility : choosing the macro will force the agent to pick-move-deliver the biscuit, while in the second case the agent could for example pick the tray and then more than one biscuit or both drink and biscuits.