UNIVERSITAT ROVIRA i VIRGILI

Waiter Robotic Task

Planning and Approximate Reasoning

João Valério

Eirik Grytøyr

Group: 16

Date: 12/11/2022

1. INTRODUCTION

In the presented report, an automated waiter robotic task will be implemented, with the primary purpose of performing the customers' order preparation and delivery.

The state world of the problem represents a restaurant with seven discrete areas: The buffet area itself (BTA), the anterior upper area (AUA), the anterior middle area (AMA), the anterior lower area (ALA), the posterior upper area (PUA), the posterior middle area (PMA), and the posterior lower area (PLA). Particularly, the areas PMA and AMA have a wall in between, indicating a non-adjacency condition.

In general, the agent, an automated waiter robot, acquires a list of customer orders containing the name and location inside the restaurant environment. For each order, the agent fills a plate with food from the BTA and then delivers it directly to the customers' table.

The final goal of the task is to find the most efficient way to accomplish a target state. To achieve this purpose, different workable solutions (domain, problem, and different initial / final states) will be implemented in PDDL. Through that, it is pretended to understand how the generalization or specialization of the implementations and the complexity of the problem affect the solution's optimality, soundness and admissibility. The parameters chosen to evaluate each implementation's optimality are the number of nodes generated, and explored, and the time complexity. The admissibility is compared through the number of steps.

2. ANALYSIS

In the development of the project, it will be considered three implementations:

a. Domain 1.

b. Domain 2.

c. Domain 3.

With the purpose of determining the optimal implementation in different initial / final states and complexities. Along the points, the generalization of the code and the capacity to adapt to distinct and close problems grow. In other words, we diminish the domain implementation, to extend the problem part, to reach the main goal of the activity. Thus, the Domain is incrementally modified, in order to comprehend how different steps influence the performance of the code.

A. Domain 1

The first domain represents the less general implementation and consists of the following predicates:

- **Robot_location(a):** The area a where the robot is.
- **location_has_customer(a):** Specify that the area a, has a customer.
- **Adjacent(a1,a2):** Area a1 and a2 are horizontally or vertically adjacent. To represent a gray wall between PMA and AMA, those should not be defined as adjacent.
- **Adjacent_BTA(a):** Area a is horizontally or vertically adjacent to BTA.
- **Served(a):** This implementation assumes that it can only be one customer for each location. To obtain fewer variables, only the location is specified as served or not.
- **robot_at_BTA():** If the robot is at BTA, it can pick up plates and fill them with food.
- **robot_holding_empty_plate:** Defines if the robot is holding an empty plate.
- **robot_holding_filled_plate:** This indicates that the robot is holding a plate filled with food.

The possible actions and the respective explanations are:

- **Pick-up()**: the robot picks up a plate, as long as it is in the BTA area. This presupposes that there are always enough plates.
- **Present(a):** The robot serves food to the customer at area a.
- **Fill():** the robot loads the empty plate with food as long as the robot is in the BTA area.
- **Move(a1,a2)**: the robot moves from area a1 to area a2.
- **Move_BTA(a):** the robot moves from area a to BTA.

The solution to this problem requires the definition of each area as an object.

- **BTA PUA AUA PMA AMA PLA ALA:** The areas are defined as the buffet area itself (BTA), the anterior upper area (AUA), the anterior middle area (AMA), the anterior lower area (ALA), the posterior upper area (PUA), the posterior middle area (PMA), and the posterior lower area (PLA).

According to the aforementioned, this is a specific implementation with a considerable amount of constraints associated. For example, the food and plates are unlimited and always situated in the BTA area. Furthermore, there is a maximum of one customer per zone. On the other hand, the advantage of this implementation is that the only type of variable and object to consider is the positions, allowing an easier modification of the problem within the constraints.

B. Domain 2

The second, slightly more general, implementation consists of the following predicates in its domain:

- **Robot_location(a):** The area a where the robot is.
- **Customer_location(c ,a):** The area a where the customer c is.
- **Plate_location(p,a)** The area a where the plate p is. In this case, it's always BTA.
- **Food_location(a):** The area where the food is. In this case, it's continuously BTA.
- **Have_Food(plate):** Defines if a plate p has food on it.
- **Served(customer):** object "customer" is served.
- **Holding(p)**: checks if the robot is holding the plate p, considering the maximum capacity as unitary.
- **Adjacent(a1,a2):** Areas a1 and a2 are horizontally or vertically adjacent. To represent a gray wall between PMA and AMA, those should not be defined as adjacent.
- **Empty_hands():** If the robot has its hands empty, it can pick up a plate.

The possible actions and the respective explanations are:

- **Pick-up(a,p)**: the robot picks up the plate p at area a.
- **Present(a, c, p):** the robot serves the plate p with food to the customer c at position a.
- **Fill(a, p):** the robot fills the empty plate p with food at area BTA.
- **Move(a1,a2)**: the robot moves from a1 to a2.

To solve the problem, the following objects are included:

- **BTA PUA AUA PMA AMA PLA ALA:** The areas are defined as: the buffet area itself (BTA), the anterior upper area (AUA), the anterior middle area (AMA), the anterior lower area (ALA), the posterior upper area (PUA), the posterior middle area (PMA), and the posterior lower area (PLA).
- **Ci:** The customers that have performed an order where "i" is the index of the customer.
- **Pi:** The plate that carries the food where "i" is the index of the plate.

In this implementation, as the plate is an object, it is necessary to consider what occurs in its position from Pick-up to Present actions. According to that, plates are objects picked at location BTA and placed in the area of the customer, assuming a new position. While the robot is holding it, the plate's position doesn't switch, until it is delivered.

This domain has one more predicate, one less action and two more types of objects than the previous one. Consequently, The possibility to define distinct customers and plates potentiates the creation of various types of problems. However, the computational cost might increase, since the algorithm has additional options to consider.

C. Domain 3

Domain 3 has the highest degree of generalization. Instead of defining the areas for each predicate or using predicates as "have_food" or "holding", a general "at" predicate is used between objects. As a consequence, it is required to define all the types of objects in the problem section. In other words, the simplicity or generalization of the domain implies a higher level of complexity in the problem definition.

The predicates in the domain are as follows:

- **Adjacent(a1,a2):** Areas a1 and a2 are horizontally or vertically adjacent. To represent a gray wall between PMA and AMA, those should not be defined as adjacent.
- **at(what ,where):** object what is located at area/object where. For instance, this can represent a robot what in an area where.
- **Served(customer):** object "customer" is served.
- **Empty_hands():** If the robot is empty, it can pick up a plate.
- **Is_robot(robot):** Defines that a parameter is a robot.
- **is_cusomer(customer):** Defines that a parameter is a customer.
- **is_plate(plate):** Defines that a parameter is a plate.
- **is_food(food):** Defines that a parameter is a food.

The possible actions and the respective explanations are:

- **Pick-up(a,p,f,r)**: the robot r picks plate p with potential food f at are a.
- **Present(a, r, p, c,f):** The robot r serves plate p with food f to customer c at the area a.
- **Fill(a, p,f,r):** The robot r fills the empty plate p with food f at the area a (BTA).
- **Move(a1,a2, r)**: The robot r moves from area a1 to area a2.

To solve the problem, the following objects are included:

- **BTA PUA AUA PMA AMA PLA ALA:** The areas are defined as: the buffet area itself (BTA), the anterior upper area (AUA), the anterior middle area (AMA), the anterior lower area

(ALA), the posterior upper area (PUA), the posterior middle area (PMA), and the posterior lower area (PLA).

- **Ci:** The customers that have performed an order, where "i" is the index of the customer.
- **Pi:** The plate that carries the food, where "i" is the index of the plate.
- **Robot**: The robot.
- **Food:** The food.

The initialization of the problem, for this implementation, requires the definition of the areas to each predicate as well as assigning objects to the "what" parameters of the predicates.

In contrast to domain 2, the objects' positions are always tracked. For example the plates change position from the BTA area, to the robot and then to the location of the customer.

The advantage of this implementation is that it increases the generalization and makes it easier to extend and modify the code. The disadvantage is that with more parameters in the actions, the number of combinations considered by the solver is higher. However, as the robot serves one customer at a time, the increment in the possibilities produces a negative effect, since the solver expands more nodes without benefiting the optimality of the path. Furthermore, it also requires more specifications of parameters in the problem.

3. PDDL IMPLEMENTATION

In the PDDL implementation the Domain is static, however, the problem changes accordingly with the initial and final states. All the problems in the study will be analyzed in the subsequent points of the report. For simplicity, for each domain, the problem considered for exemplification in Appendix B is the one proposed in the assignment (problem 2 in table 1). However, all the code is developed in the PDDL files according to the considerations explained in the present document. The Problems in the file are enumerated <domain number>-<problem number>.

4. TESTING CASES

In order to test the different implementations described above, five problems with increasing complexity was considered as shown in table 1.

Table 1 - Initial and final states of the five problems.

	Initial State		Final State	
Problem 1	■	🤖🍽️🍔	■	🤖🍽️🍔
		🙎		🙎🍔🍽️
Problem 2*	■	🤖🍽️🍔	■	🤖
	🙎		🙎🍔🍽️	
Problem 3	■	🤖🍽️🍽️🍔🍔	■	🤖
	🙎	🙎	🙎🍔🍽️	🙎🍔🍽️
Problem 4	■	🍽️🍽️🍽️🍔🍔🍔🤖	■	🤖
	🙎	🙎	🙎🍔🍽️	🙎🍔🍽️
		🙎		🙎🍔🍽️
Problem 5	■	🍽️🍽️🍽️🍽️🍽️🍽️🍔 🍔🍔🍔🍔🍔🤖	■	
	🙎	🙎	🙎🍔🍽️	🙎🍔🍽️
	🙎	🙎	🙎🍔🍽️	🙎🍔🍽️
	🙎	🙎	🙎🍔🍽️🤖	🙎🍔🍽️

* The initial and final states proposed in the exercise.

Labels: 🤖 - robot, 🍽️ - plate, 🍔 - food and 🙎 - customer.

From problems 1 to 4, the initial position of the robot is static, since it is pretended to comprehend the effects of the number of orders and the wall between PMA and AMA. In problem 5, it is considered the maximum complexity, where all the areas have one customer requiring service and the robot ends in the farthest position from BTA. Finally, all the domains function correctly if the robot has to pick up empty or already-filled plates, as it was properly tested. However, in the initial state, the plates are consistently considered empty, in order to test the most complex cases.

5. RESULT AND ANALYSIS

The performance results for each domain of the five problems considered are presented in the tables below, enumerated from 2 to 5. Moreover, the steps taken by the robot are presented in appendix A, while the path is demonstrated in chapter 6.

Table 2 represents the number of steps or actions performed in the plan considered by the robot to reach the solution.

Table 2 – Number of steps.

	Problem 1	Problem 2	Problem 3	Problem 4	Problem 5
Domain 1	7	9	20	29	48
Domain 2	7	9	16	25	46
Domain 3	7	9	16	27	46

According to the values presented in table 2, it is observable that the number of steps needed to reach the final state increase along with the problem complexity, represented by the number of customers.

Relatively to the least complex problems (1 and 2), independently of the domain, the solutions achieved are similar and correspond to the most optimal path. However, when the complexity increases, the path defined by the three domains differs. In problem 3, domains 2 and 3 can accomplish the optimal path (16 steps) to reach the solution, whereas, in problem 4, the second domain is the only one that reaches the goal in an optimal way (25 steps). Finally, in the most complex problem 5, the solutions from domain 1 are not optimal. This is probably due to the level of generalization of the resolution, in which more combinations considered can lead to a higher probability of a better solution selected. In this case, 2 extra steps were taken for domain 1.

The following table exposes the number of nodes generated during the search process.

Table 3 – Number of nodes generated.

	Problem 1	Problem 2	Problem 3	Problem 4	Problem 5
Domain 1	29	38	82	118	200
Domain 2	32	42	101	189	493
Domain 3	32	42	88	185	472

According to table 3, the increment of complexity implies a growth rate in the number of nodes generated, which follows the same pattern as in table 2.

Overall, the number of nodes generated is consistently less in domain 1, the most problem-specific resolution, because with a lower number of combinations considered, there are fewer nodes to be generated. Furthermore, the difference in the number of nodes from domain 1 to the remaining ones increases along with the complexity of the problem. Between domains 2 and 3, the number of nodes generated only differs from problems 3 to 5 (highest complexity), in which the latter always generates fewer nodes.

According to the previous analysis, the dominant pattern indicates that with fewer generated nodes, the options considered are lower, possibly leading to sub-optimal paths, especially in increased complexity problems.

Table 4 is relative to the number of nodes expanded in each problem.

Table 4 – Number of nodes expanded.

	Problem 1	Problem 2	Problem 3	Problem 4	Problem 5
Domain 1	21	29	72	103	178
Domain 2	20	31	40	71	175
Domain 3	20	31	35	72	175

The number of nodes expanded in table 4 gives an understanding of which nodes generated (table 3) were considered as the most valid at each level of the tree. As shown by the table values, the difference between domains in the same problems is not as large as in table 3. Thus, considering tables 2, 3 and 4, it is inferable that all the domains achieve optimal or satisfactory sub-optimal paths to reach the solution, in which the difference is mainly in the number of nodes generated (table 3). In fact, the most dramatic discrepancies in the nodes expanded (intermediate complexity problems 3 and 4) are not significant.

Between resolutions, domain 1 only outperforms the remaining in problem 2, while domains 2 and 3 represent extremely similar behaviours, being the highest discrepancy of only 5 nodes. Furthermore, it is important to denote that the differences registered on more complex problems need to have more importance in the final considerations.

The table, compared with the number of steps generates, shows a higher correlation between the number of nodes expanded and the number of steps in the solution, than the complexity of the problem.

The time consumption, represented in table 5, is the last metric to evaluate.

Table 5 – Time consumption in seconds.

	Problem 1	**Problem 2**	**Problem 3**	**Problem 4**	**Problem 5**
Domain 1	0.268	0.269	0.263	0301	0.382
Domain 2	0.287	0.268	0.304	0.294	0.390
Domain 3	0.274	0.256	0.270	0.268	0.374

In the last table, the time consumption is the metric to be evaluated. In general, it is inferable that time increases with the complexity of the problem, because more actions need to be taken and, therefore, more nodes are generated.

The time of each cell is the median value of 4 repeated tests, to compensate for the inference of load on the CPU. However, the values are not significantly distinguishable between the domains, indicating that this metric does not point out the most satisfactory domain(s).

6. OUTPUT PATH

All the actions of the output path are displayed in appendix A and visually represented in tables 6 to 9. The tables illustrate the initial state for the placement of objects, along with the step order conducted by the robot, in which each delivery performed by the robot is represented with a distinct color.

Table 6 is related to problems 1 and 2 for all the domains.

Table 6 – Output path for problems 1 and 2.

Domains 1, 2 and 3			
Problem 1		Problem 2	
⬛	🤖🍽️🍔 0,4	⬛	🤖🍽️🍔 0,6
	1,3	2,4	1,5
	2 🙎	3 🙎	

Table 6 shows that the output path for the least complex situations is completely optimal in all the domains since the robot is moving the shortest distance with the food from BAT to the position of the customers and back.

The table illustrates problem 3 for all the domains.

Table 7 – Output path for problem 3.

Problem 3					
Domain 1		Domains 2		Domains 3	
⬛	🤖🍽️🍽️ 🍔🍔 0,8,14	⬛	🤖🍽️🍽️ 🍔🍔 0,4,10	⬛	🤖🍽️🍽️ 🍔🍔 0,6,10
6,10,12	1,7,9,13	6,8	1,3,5,9	2,4,6,8	1,5,7,9
5,11 🙎	2 🙎	7 🙎	2 🙎	3 🙎	8 🙎
4	3				

According to the previous table, it is possible to confirm the analysis developed priorly, indicating that domains 2 and 3 achieve the best optimal path, while domain 1 is not able to accomplish it. However, it is essential to denote that the steps performed by the robot in domains 2 and 3 are distinct, indicating that for a specific problem more than 1 optimal path might be valid. Finally, in domain 1 between steps 2 and 3, the robot instead of moving to AUA, goes down to ALA. Consequently, a suboptimal solution is achieved.

Table 8 represents problem 4 for each domain.

Table 8 – Output path for problem 4.

Problem 4					
Domain 1		**Domain 2**		**Domain 3**	
⬛	🍽️🍽️🍽️ 🍔🍔🍔🤖 0,8,14,20	⬛	🍽️🍽️🍽️ 🍔🍔🍔🤖 0,4,10,16	⬛	🍽️🍽️🍽️ 🍔🍔🍔🤖 0,4,12,18
6,16,18	1,7,9,13,15,19	6,8	1,3,5,9,11,15	6	1,3,5,11,13,17
5,**17** 🙎	**2**,10,12 🙎	**7** 🙎	**2**,12,14 🙎	**7** 🙎	**2**,10,12,14,16 🙎
4	3,**11** 🙎		**13** 🙎	8	**9**, **15** 🙎

Relatively to problem 4, only domain 2 is able to reach the optimal solution since the remaining take a few extra steps due to the node selection in a certain stage. Particularly, in domain 3 the difference is essentially in the trajectory executed to get the food to deliver to the third customer (Green), where 2 extra steps are taken. In domain 1, 4 extra steps are performed while serving the second customer (blue). In both domains (1 and 3), the robot instead of moving back to BAT directly (upward direction in the table), decides to move in a downward trajectory. The reason for the detour is caused by how the algorithm chooses the nodes to expand, perhaps by trying to avoid entering the same state multiple times.

Finally, table 9 demonstrates the path taken by the robot in problem 5.

Table 9 – Output path for problem 5.

Problem 5			
Domain 1		**Domains 2 and 3**	
▨	🍽️🍽️🍽️🍽️🍽️🍽️🍔🍔🍔🍔 🍔🍔🤖 0,2,6,14,20,26	▨	🍽️🍽️🍽️🍽️🍽️🍽️🍔🍔🍔 🍔🍔🍔🤖 0,2,6,10,18,24
8,**22**,**24** 🙅	**1**,3,5,7,13,15,19,21,25,27 🙅	**8**,**16**,**20**,**22**,**26** 🙅	**1**,3,5,7,9,11,17,19,23,25 🙅
9,**23** 🙅	**4**,**12**,**16**,18,28 🙅	15,**21**,**27** 🙅	**4**,**12**,18 🙅
10,**30** 🙅	11,**17**,**29** 🙅	14,**28** 🙅	**13** 🙅

Table 9 shows the path for the robot in problem 5, where domains 2 and 3 output the same shortest path, while domain 1 adds extra steps. However, it is important to denote that none of the domains is capable of reaching the optimal solution. In Domain 1, in the fourth customer (yellow) serving procedure, the robot assumes 4 extra steps by moving to PLA instead of PUA or AMA. In domains 2 and 3, when the robot needs to serve the fifth customer (pink), instead of moving upward in the table to AMA, takes a longer path by entering the PLA area.

According to tables 7, 8 and 9, a pattern is displayed, in which the trajectory from the BTA to the customer is consistently the shortest distance, but in the opposite course, the robot, circumstantially, performs sub-optimal decisions.

By converging chapters 5 and 6 analysis, it is inferable that the best domains for this overall study are 2 and 3, the most general approaches. In general, domain 2 is the most satisfactory because it reaches the optimal path in 4 of the 5 problems faced. However, even though domain 3 reaches the optimal solution in 3 problems, in the remaining ones the paths found are sub-optimal, which also indicates a suitable solution. Furthermore, this domain might be preferable if it is pretended to diminish the number of nodes expanded in complex problems, assuming that a sub-optimal solution would be sufficient.

7. CONCLUSIONS

All the goals initially proposed were successfully achieved and they will be explained in the conclusions:

Firstly, it was understandable that for a specific problem, many different solutions are possible to be implemented, with various levels of generalization.

Moreover, the metrics chosen to evaluate the models demonstrate a one-way relationship with the increase or decrease of complexity. Besides that, it was observable that a strong association between the number of steps and the nodes expanded, stating that the search tree considered by the robot is a representation of the environment. In terms of time, along the problems, the increment is not relevant and between the domains, it was not possible to point out a distinction.

It is important to denote that, for this implementation, the overall best solutions correspond to domains 2 and 3, which are characterized as general. In terms of optimality, the second domain is most satisfactory, while the third might be preferable when less memory consumption is needed since the trade-off between optimality and memory is acceptable. Furthermore, domain 1 would only be a choice in extremely low complexity levels, which is not suitable for real-world applications.

Additionally, the test shows that the PDDL planner algorithm for the tested cases is not admissible, but sound and in the tested problems it is complete. The optimality depends heavily on how the actions and predicates are defined. The more specific the domain is, the fewer nodes are generated and might lead to fewer options considered. In the case with this solver, Domain 1 found a solution with few nodes generated, but it was often not the optimal solution.

To finish, it is concluded that for simple problems like the one proposed in this study, it is conceivable to infer that the majority of solutions, at any level of generalization, are able to accomplish suitable results.

Appendix A: Planer outputs

The names of the areas are shown in table A1:

Table 1 - The name of the rooms used by the solver.

▨	Bta
Pua	Aua
Pma	Ama
Pla	Ala

Problem 1:

domain 1	domain 2	domain 3
0.00100: (pick_up)	0.00100: (pick_up bta p1)	0.00100: (pick_up bta p1 food robot)
0.00200: (fill)	0.00200: (fill bta p1)	0.00200: (fill bta p1 food robot)
0.00300: (move_area ala aua)	0.00300: (move bta aua)	0.00300: (move bta aua robot)
0.00400: (move_area aua ama)	0.00400: (move aua ama)	0.00400: (move aua ama robot)
0.00500: (present ama)	0.00500: (present ama c1 p1)	0.00500: (present ama robot p1 c1 food)
0.00600: (move_area ama aua)	0.00600: (move ama aua)	0.00600: (move ama aua robot)
0.00700: (move_bta aua)	0.00700: (move aua bta)	0.00700: (move aua bta robot)

Problem 2:

domain 1	domain 2	domain 3
0.00100: (pick_up)	0.00100: (pick_up bta p1)	0.00100: (pick_up bta p1 food robot)
0.00200: (fill)	0.00200: (fill bta p1)	0.00200: (fill bta p1 food robot)
0.00300: (move_area ala aua)	0.00300: (move bta aua)	0.00300: (move bta aua robot)
0.00400: (move_area aua pua)	0.00400: (move aua pua)	0.00400: (move aua pua robot)
0.00500: (move_area pua pma)	0.00500: (move pua pma)	0.00500: (move pua pma robot)
0.00600: (present pma)	0.00600: (present pma c1 p1)	0.00600: (present pma robot p1 c1 food)
0.00700: (move_area pma pua)	0.00700: (move pma pua)	0.00700: (move pma pua robot)
0.00800: (move_area pua aua)	0.00800: (move pua aua)	0.00800: (move pua aua robot)
0.00900: (move_bta aua)	0.00900: (move aua bta)	0.00900: (move aua bta robot)

Problem 3:

domain 1	domain 2	domain 3
0.00100: (pick_up)	0.00100: (pick_up bta p2)	0.00100: (pick_up bta p1 food robot)
0.00200: (fill)	0.00200: (fill bta p2)	0.00200: (fill bta p1 food robot)
0.00300: (move_area ala aua)	0.00300: (move bta aua)	0.00300: (move bta aua robot)
0.00400: (move_area aua ama)	0.00400: (move aua ama)	0.00400: (move aua pua robot)
0.00500: (present ama)	0.00500: (present ama c2 p2)	0.00500: (move pua pma robot)
0.00600: (move_area ama ala)	0.00600: (move ama aua)	0.00600: (present pma robot p1 c1 food)
0.00700: (move_area ala pla)	0.00700: (move aua bta)	0.00700: (move pma pua robot)
0.00800: (move_area pla pma)	0.00800: (pick_up bta p1)	0.00800: (move pua aua robot)
0.00900: (move_area pma pua)	0.00900: (fill bta p1)	0.00900: (move aua bta robot)
0.01000: (move_area pua aua)	0.01000: (move bta aua)	0.01000: (pick_up bta p2 food robot)
0.01100: (move_bta aua)	0.01100: (move aua pua)	0.01100: (fill bta p2 food robot)
0.01200: (pick_up)	0.01200: (move pua pma)	0.01200: (move bta aua robot)
0.01300: (fill)	0.01300: (present pma c1 p1)	0.01300: (move aua ama robot)
0.01400: (move_area ala aua)	0.01400: (move pma pua)	0.01400: (present ama robot p2 c2 food)

0.01500: (move_area aua pua)	0.01500: (move pua aua)	0.01500: (move ama aua robot)
0.01600: (move_area pua pma)	0.01600: (move aua bta)	0.01600: (move aua bta robot)
0.01700: (present pma)		
0.01800: (move_area pma pua)		
0.01900: (move_area pua aua)		
0.02000: (move_bta aua)		

Problem 4:

domain 1	domain 2	domain 3
0.00100: (pick_up)	0.00100: (pick_up bta p3)	0.00100: (pick_up bta p1 food robot)
0.00200: (fill)	0.00200: (fill bta p3)	0.00200: (fill bta p1 food robot)
0.00300: (move_area ala aua)	0.00300: (move bta aua)	0.00300: (move bta aua robot)
0.00400: (move_area aua ama)	0.00400: (move aua ama)	0.00400: (move aua ama robot)
0.00500: (present ama)	0.00500: (present ama c2 p3)	0.00500: (present ama robot p1 c2 food)
0.00600: (move_area ama ala)	0.00600: (move ama aua)	0.00600: (move ama aua robot)
0.00700: (move_area ala pla)	0.00700: (move aua bta)	0.00700: (move aua bta robot)
0.00800: (move_area pla pma)	0.00800: (pick_up bta p3)	0.00800: (pick_up bta p3 food robot)
0.00900: (move_area pma pua)	0.00900: (fill bta p3)	0.00900: (fill bta p3 food robot)
0.01000: (move_area pua aua)	0.01000: (move bta aua)	0.01000: (move bta aua robot)
0.01100: (move_bta aua)	0.01100: (move aua pua)	0.01100: (move aua pua robot)
0.01200: (pick_up)	0.01200: (move pua pma)	0.01200: (move pua pma robot)
0.01300: (fill)	0.01300: (present pma c1 p3)	0.01300: (present pma robot p3 c1 food)

0.01400: (move_area ala aua)	0.01400: (move pma pua)	0.01400: (move pma pla robot)
0.01500: (move_area aua ama)	0.01500: (move pua aua)	0.01500: (move pla ala robot)
0.01600: (move_area ama ala)	0.01600: (move aua bta)	0.01600: (move ala ama robot)
0.01700: (present ala)	0.01700: (pick_up bta p3)	0.01700: (move ama aua robot)
0.01800: (move_area ala ama)	0.01800: (fill bta p3)	0.01800: (move aua bta robot)
0.01900: (move_area ama aua)	0.01900: (move bta aua)	0.01900: (pick_up bta p2 food robot)
0.02000: (move_bta aua)	0.02000: (move aua ama)	0.02000: (fill bta p2 food robot)
0.02100: (pick_up)	0.02100: (move ama ala)	0.02100: (move bta aua robot)
0.02200: (fill)	0.02200: (present ala c3 p3)	0.02200: (move aua ama robot)
0.02300: (move_area ala aua)	0.02300: (move ala ama)	0.02300: (move ama ala robot)
0.02400: (move_area aua pua)	0.02400: (move ama aua)	0.02400: (present ala robot p2 c3 food)
0.02500: (move_area pua pma)	0.02500: (move aua bta)	0.02500: (move ala ama robot)
0.02600: (present pma)		0.02600: (move ama aua robot)
0.02700: (move_area pma pua)		0.02700: (move aua bta robot)

Problem 5:

domain 1	domain 2	domain 3
0.00100: (pick_up)	0.00100: (pick_up bta p6)	0.00100: (pick_up bta p1 food robot)
0.00200: (fill)	0.00200: (fill bta p6)	0.00200: (fill bta p1 food robot)
0.00300: (move_area ala aua)	0.00300: (move bta aua)	0.00300: (move bta aua robot)
0.00400: (present aua)	0.00400: (present aua c4 p6)	0.00400: (present aua robot p1 c5 food)
0.00500: (move_bta aua)	0.00500: (move aua bta)	0.00500: (move aua bta robot)
0.00600: (pick_up)	0.00600: (pick_up bta p5)	0.00600: (pick_up bta p6 food robot)
0.00700: (fill)	0.00700: (fill bta p5)	0.00700: (fill bta p6 food robot)
0.00800: (move_area ala aua)	0.00800: (move bta aua)	0.00800: (move bta aua robot)
0.00900: (move_area aua ama)	0.00900: (move aua ama)	0.00900: (move aua ama robot)
0.01000: (present ama)	0.01000: (present ama c2 p5)	0.01000: (present ama robot p6 c2 food)
0.01100: (move_area ama aua)	0.01100: (move ama aua)	0.01100: (move ama aua robot)
0.01200: (move_bta aua)	0.01200: (move aua bta)	0.01200: (move aua bta robot)
0.01300: (pick_up)	0.01300: (pick_up bta p4)	0.01300: (pick_up bta p5 food robot)
0.01400: (fill)	0.01400: (fill bta p4)	0.01400: (fill bta p5 food robot)
0.01500: (move_area ala aua)	0.01500: (move bta aua)	0.01500: (move bta aua robot)
0.01600: (move_area aua pua)	0.01600: (move aua pua)	0.01600: (move aua pua robot)
0.01700: (present pua)	0.01700: (present pua c6 p4)	0.01700: (present pua robot p5 c6 food)

0.01800: (move_area pua pma)	0.01800: (move pua aua)	0.01800: (move pua aua robot)
0.01900: (move_area pma pla)	0.01900: (move aua bta)	0.01900: (move aua bta robot)
0.02000: (move_area pla ala)	0.02000: (pick_up bta p3)	0.02000: (pick_up bta p4 food robot)
0.02100: (move_area ala ama)	0.02100: (fill bta p3)	0.02100: (fill bta p4 food robot)
0.02200: (move_area ama aua)	0.02200: (move bta aua)	0.02200: (move bta aua robot)
0.02300: (move_bta aua)	0.02300: (move aua ama)	0.02300: (move aua ama robot)
0.02400: (pick_up)	0.02400: (move ama ala)	0.02400: (move ama ala robot)
0.02500: (fill)	0.02500: (present ala c3 p3)	0.02500: (present ala robot p4 c3 food)
0.02600: (move_area ala aua)	0.02600: (move ala pla)	0.02600: (move ala pla robot)
0.02700: (move_area aua ama)	0.02700: (move pla pma)	0.02700: (move pla pma robot)
0.02800: (move_area ama ala)	0.02800: (move pma pua)	0.02800: (move pma pua robot)
0.02900: (present ala)	0.02900: (move pua aua)	0.02900: (move pua aua robot)
0.03000: (move_area ala ama)	0.03000: (move aua bta)	0.03000: (move aua bta robot)
0.03100: (move_area ama aua)	0.03100: (pick_up bta p2)	0.03100: (pick_up bta p3 food robot)
0.03200: (move_bta aua)	0.03200: (fill bta p2)	0.03200: (fill bta p3 food robot)
0.03300: (pick_up)	0.03300: (move bta aua)	0.03300: (move bta aua robot)
0.03400: (fill)	0.03400: (move aua pua)	0.03400: (move aua pua robot)
0.03500: (move_area ala aua)	0.03500: (move pua pma)	0.03500: (move pua pma robot)

0.03600: (move_area aua pua)	0.03600: (present pma c1 p2)	0.03600: (present pma robot p3 c1 food)
0.03700: (move_area pua pma)	0.03700: (move pma pua)	0.03700: (move pma pua robot)
0.03800: (present pma)	0.03800: (move pua aua)	0.03800: (move pua aua robot)
0.03900: (move_area pma pua)	0.03900: (move aua bta)	0.03900: (move aua bta robot)
0.04000: (move_area pua aua)	0.04000: (pick_up bta p1)	0.04000: (pick_up bta p2 food robot)
0.04100: (move_bta aua)	0.04100: (fill bta p1)	0.04100: (fill bta p2 food robot)
0.04200: (pick_up)	0.04200: (move bta aua)	0.04200: (move bta aua robot)
0.04300: (fill)	0.04300: (move aua ama)	0.04300: (move aua ama robot)
0.04400: (move_area ala aua)	0.04400: (move ama ala)	0.04400: (move ama ala robot)
0.04500: (move_area aua ama)	0.04500: (move ala pla)	0.04500: (move ala pla robot)
0.04600: (move_area ama ala)	0.04600: (present pla c5 p1)	0.04600: (present pla robot p2 c4 food)
0.04700: (move_area ala pla)		
0.04800: (present pla)		

Appendix B: The code

A. Domain 1 and Problem 2

```
;The state world of the problem representing a building including 7 areas in a restaurant. One of them are the BTA area which is the kitchen,

;The BTA area consosts of unlimited plates and food. the other areas can contain up to one customer each.

;The robot needs to pick up a plate, fill it with food, and give it to a customer on a specified place.


(define (domain Waiter_domain_1)

  (:requirements :strips :adl)

  (:predicates

    (Robot_location ?a)

    (location_has_customer ?a)

    (Served ?a)

    (robot_at_BTA)

    (robot_holding_empty_plate)

    (robot_holding_filled_plate)

    (Adjacent ?a1 ?a2)

    (Adjacent_BTA ?a)

  )


  ;The robot moves from area a1 to area a2.

  (:action move_area

    :parameters (?a1 ?a2)

    :precondition (or

      (and (Robot_location ?a1) (or (Adjacent ?a1 ?a2) (Adjacent ?a2 ?a1)))

      (and (Adjacent_BTA ?a2) (robot_at_BTA)) )

    :effect (and (not (Robot_location ?a1))
      (and (robot_location ?a2) (not (robot_at_BTA))))

  )

  ;the robot moves from area a to BTA

  (:action move_BTA

    :parameters (?a)

    :precondition (and

      (Robot_location ?a) (Adjacent_BTA ?a))

    :effect (and (not (Robot_location ?a))

      (robot_at_BTA) )

  )

  ;The robot serves food to the customer at area a

  (:action Present

    :parameters (?a)

    :precondition (and

      (Robot_location ?a) (location_has_customer ?a)
(robot_holding_filled_plate) )

    :effect (and (Served ?a) (not (robot_holding_filled_plate)) (not
(robot_at_BTA)))

    )


  ; the robot picks up an empty plate, as long as it is in the BTA area.

  ;This  presupposes that there are always enough plates.

    (:action Pick_up

    :parameters ()

    :precondition (

      robot_at_BTA )

    :effect (and (robot_holding_empty_plate))

    )

    ;the robot serves the plate with food as long as it is in the BTA area.
    (:action Fill

    :parameters ()

    :precondition (and

      (robot_at_BTA) (robot_holding_empty_plate))

    :effect (and (robot_holding_filled_plate) (not(robot_holding_empty_plate)))

      )

)
```

Problem

```
;The state world of the problem representing a building including 7 areas in a restaurant. One of them are the BTA area which is the kitchen,

;The BTA area consosts of unlimited plates and food. the other areas can contain up to one customer each.

;The robot needs to pick up a plate, fill it with food, and give it to a customer on a specified place.



(define (problem Waiter_1-2)

  (:domain Waiter_domain_1)

  (:objects

    BTA PUA AUA PMA AMA PLA ALA

  )

  (:init

    ;(Robot_location AMA)

    (robot_at_BTA)

    ;Define location of each customer

    (location_has_customer PMA)

    ;(location_has_customer AUA)

    ;(location_has_customer AMA)

    ;(location_has_customer ALA)
    ;(location_has_customer PLA)

    ;(location_has_customer PUA)


    ;Define which room that is adjacent.

    (Adjacent PUA AUA)

    (Adjacent PMA PUA)

    (Adjacent PLA PMA)

    (Adjacent AMA AUA)

    (Adjacent PLA ALA)

    (Adjacent ALA AMA)

    (Adjacent_BTA AUA)


  )

  (:goal

    (and (robot_at_BTA)

     (Served PMA) ;(Served AUA) ;(Served PUA)

     ;(Served PLA) (Served ALA) (Served AMA)

    )

  )

)
```

B. Domain 2 and Problem 2

```
;The state world of the problem representing a building including 7 areas in a restaurant. One of them are the BTA area which is the kitchen,

;The BTA area consosts of plates and unlimited food. the other areas can contain customers.

;The robot needs to pick up a defined plate, fill it with food, and give it to a specified customer on a specified place.


(define (domain Waiter_domain_2)

  (:requirements :strips :adl)

  (:predicates

    (Robot_location ?a)

    (Customer_location ?c ?a)

    (Served ?Customer)

    (Plate_location ?p ?a)

    (Have_food ?plate)

    (Adjacent ?a1 ?a2)

    (Holding ?p)

    (Empty_hands)

    (Food_location ?a)

  )

  ;The robot moves from area a1 to area a2.

  (:action move

    :parameters (?a1 ?a2)

    :precondition (and

      (Robot_location ?a1) (or (Adjacent ?a1 ?a2) (Adjacent ?a2 ?a1) ))

    :effect (and (not (Robot_location ?a1))

      (robot_location ?a2) )

  )
  ;The robot serves the plate p of food to the customer c at position a.

  (:action Present

    :parameters (?a ?c ?p)

    :precondition (and

      (Robot_location ?a) (Customer_location ?c ?a) (have_food ?p) (holding ?p)
)

    :effect (and (Served ?c) (Plate_location ?p ?a ) (Empty_hands) (not
(holding ?p)) (not(have_food ?p)))

  )

    ;The robot picks up the plate p at area a. To make sure that the robot is
not picking up served plates, and giving it to another customer it will not
pick up plates with food.


    (:action Pick_up

    :parameters (?a ?p)

    :precondition (and

      (Robot_location ?a) (Plate_location ?p ?a ) (Empty_hands))

    :effect (and (not(plate_location ?a ?p)) (not (Empty_hands )) (Holding ?p)
)

    )

      ;The robot fills the plate p with food at area BTA.

    (:action Fill

    :parameters (?a ?p)

    :precondition (and

      (Robot_location ?a) (Food_location ?a) (holding ?p))

    :effect (Have_food ?p)

      )

)
```

Problem

```
;The state world of the problem representing a building including 7 areas
in a restaurant. One of them are the BTA area which is the kitchen,

;The BTA area consosts of plates and unlimited food. the other areas can
contain customers.

;The robot needs to pick up a defined plate, fill it with food, and give
it to a specified customer on a specified place.


(define (problem Waiter_2-2)

  (:domain Waiter_domain_2)

  (:objects

    BTA PUA AUA PMA AMA PLA ALA

    P1 ;P2 ;P3 P4 P5 P6

    C1 ;C2 ;C3 C4 C5 C6

  )

  (:init

    (Robot_location BTA)


    (Empty_hands)

    (Food_location BTA)


    ;Define location of each plate

    (Plate_location P1 BTA)

    ;(Plate_location P2 BTA)

    ;(Plate_location P3 BTA)

    ;(Plate_location P4 BTA)

    ;(Plate_location P5 BTA)

    ;(Plate_location P6 BTA)
```

```
          ;Define location of each customer

          (Customer_location C1 PMA)

          ;(Customer_location C2 PMA)

          ;(Customer_location C3 AUA)

          ;(Customer_location C4 ALA)

          ;(Customer_location C5 PLA)

          ;(Customer_location C6 PUA)


          ;Define which room that is adjacent.

          (Adjacent BTA AUA)

          (Adjacent PUA AUA)

          (Adjacent PMA PUA)

          (Adjacent PLA PMA)

          (Adjacent AMA AUA)

          (Adjacent PLA ALA)

          (Adjacent ALA AMA)


        )

        (:goal

          (and (Robot_location BTA)

           (Served C1) ;(Served C2) ;(Served C3)

           ;(Served C4) (Served C5) (Served C6)

          )


      )

      )
```

C. Domain 3 and Problem 2

```
;The state world of the problem representing a building including 7 areas in a restaurant. One of them are the BTA area which is the kitchen,

;The BTA area consosts of plates and unlimited food. the other areas can contain customers.

;The robot needs to pick up a defined plate, fill it with food, and give it to a specified customer on a specified place.


(define (domain Waiter_domain_3)

  (:requirements :adl :typing)

  (:predicates


    (at ?what ?where)

    (Served ?Customer)


    (Is_robot ?robot)

    (is_customer ?customer)

    (is_plate ?plate)

    (is_food ?food)

    (Adjacent ?a1 ?a2)


    (Empty_hands)


  )


  ;The robot r moves from area a1 to area a2.

  (:action move

    :parameters (?a1 ?a2 ?r)

    :precondition (and
      (at ?r ?a1) (or (Adjacent ?a1 ?a2) (Adjacent ?a2 ?a1)) (Is_robot ?r))

    :effect (and (not (at ?r ?a1))

      (at ?r ?a2) )

  )

  ;The robot r serves plate p with food f to customer c at area a

  (:action Present

    :parameters (?a ?r ?p ?c ?f)

    :precondition (and

      (at ?r ?a) (at ?c ?a) (at ?p ?r) (at ?f ?p) (Is_robot ?r) (is_customer
?c) (is_plate ?p) (is_food ?f))

    :effect (and (Served ?c) (at ?p ?a) (not (at ?p ?r)) (Empty_hands) (not(at
?f ?p)) )

      )

    ;The robot r picks plate p if it has not food f at are a.


    (:action Pick_up

    :parameters (?a ?p ?f ?r)

    :precondition (and

      (at ?r ?a) (at ?p ?a) (Empty_hands) (is_plate ?p) (is_food ?f) (Is_robot
?r))

    :effect (and (not(at ?p ?a)) (not (Empty_hands )) (at ?p ?r))

    )

    ;The robot r fills the plate p with food f at area a = BTA.

    (:action Fill

    :parameters (?a ?p ?f ?r)

    :precondition (and

      (at ?r ?a) (at ?f ?a) (at ?p ?r) (Is_robot ?r) (is_plate ?p) (is_food
?f))

    :effect (at ?f ?p)

      )
```

```
)
```

Problem

```
;The state world of the problem representing a building including 7 areas in a restaurant. One of them are the BTA area which is the kitchen,

;The BTA area consosts of plates and unlimited food. the other areas can contain customers.

;The robot needs to pick up a defined plate, fill it with food, and give it to a specified customer on a specified place.


(define (problem Waiter_3-2)

  (:domain Waiter_domain_3)

  (:objects

    BTA PUA AUA PMA AMA PLA ALA

    P1 ;P2 ;P3 P4 P5 P6

    C1 ;C2 C3; C4 C5 C6

    Robot

    Food

  )

  (:init

    (at Robot BTA)


    (Empty_hands)

    (at Food BTA)


    ;Define location of each plate

    (at P1 BTA)

    ;(at P2 BTA)

    ;(at P3 BTA)

    ;(at P4 BTA)
```

```
;(at P5 BTA)

;(at P6 BTA)




;Define location of each customer

(at C1 PMA)

;(at C2 AMA)

;(at C3 ALA)

;(at C4 PLA)

;(at C5 AUA)

;(at C6 PUA)


;Define which room that is adjacent.

(Adjacent BTA AUA)

(Adjacent PUA AUA)

(Adjacent PMA PUA)

(Adjacent PLA PMA)

(Adjacent AMA AUA)

(Adjacent PLA ALA)

(Adjacent ALA AMA)


;Is stuff

(is_customer C1)

;(is_customer C2)

;(is_customer C3)

;(is_customer C4)

;(is_customer C5)

;(is_customer C6)
```

```
    (is_plate P1)
    ;(is_plate P2)
    ;(is_plate P3)
    ;(is_plate P4)
    ;(is_plate P5)
    ;(is_plate P6)
    (Is_robot Robot)
    (Is_food Food)



  )
  (:goal
    (and (at Robot BTA)
     (Served C1) ;(Served C3); (Served C3)
     ;(Served C4) (Served C5) (Served C6)
     ;(at P1 Robot)
    )
  )
)
```