UNIVERSITAT ROVIRA i VIRGILI

Cleaner Robotic Task

Planning and Approximate Reasoning

João Valério

Eirik Grytøyr

Group: 16

Date: 15/10/2022

1. INTRODUCTION

In the presented report, an automated cleaner robotic task will be implemented, with the main purpose of performing the cleaning of the building considered.

The state world of the problem representing the building includes nine offices, between zero and eight boxes and one robot, which is the task performer. Particularly, each office is defined by two combinations of states (cleaned or dirty and empty or not empty). The robot can move vertically or horizontally between adjacent offices. Furthermore, it can move one box to an adjacent empty office.

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 affect the efficiency of the solution. The parameters chosen to evaluate the optimality of each implementation are the number of nodes generated, explored, the time complexity and number of steps.

2. ANALYSIS

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

a. Domain 1.
b. Domain 2.
c. Domain 3.
d. Domain 4.

With the purpose of determining the optimal implementation in different initial / final states. Along the points the generalization of the code, or capacity to be applied to distinct and close problems, grows. In other words, we diminish the domain implementation in, to extend the problem part, to reach the main goal of the activity. Each Domain is incrementally changed from the previous one, to be able to see how different steps change the performance of the code.

A. Domain 1

The first, and less general, implementation that will be considered consists of the following predicates in its domain:

- **Robot-location(o):** the robot is in office o.
- **Box-location(A,o):** box A is located in office o.
- **Dirty(o):** office o is dirty.
- **Clean(o):** office o is clean.
- **Empty(o):** there isn't any box in office o.
- **Adjacent(o1,o2):** offices o1 and o2 are horizontally or vertically adjacent.

The possible actions and the respective explanations are:

- **Clean-office(o)**: the robot cleans office o.
- **Move(o1,o2)**: the robot moves from office o1 to office o2.
- **Push(A,o1,o2)**: the robot pushes box A from office o1 to office o2.

To solve the problem, the following objects are included:

- **oi:** The office, where "i" is the office index. The number of offices goes from 1 to 9.
- **bi:** The boxes, where "i" is the box index. The number of boxes is between 0 and 8.

The initialization of the problem, for this implementation, requires the definition of the office to each predicate and the boxes considered. This way, this is the most general implementation.

B. Domain 2

The second solution tries to increase the optimality from domain 1, without changing the level of generalization of the implementation. Therefore, the only distinction is the fact that the Clean(o) predicate is deleted, since Dirty(o) is enough to describe the office state, due to its binary behavior. In other words, if Dirty(o) is true, the office 'o' is dirty, whereas Dirty(o) is false, means it is clean.

C. Domain 3

In the third solution, the generalization is increased, by simplifying the domain, with the cost of an increased complexity on the problem definition. So, it is considered a distinct way of defining the adjacent offices, compared to the previous proposals. In the action of domains 1 and 2, the adjacency between two offices is verified in both directions recurring to an 'or' condition (a adjacent to b or b adjacent to a). This way, the initialization of the problem is simpler.

In domain 3, the conditions related to the adjacent offices are reduced to one. Consequentially, the number of initializations related to the adjacent offices increases by double. The possible advantage of this approach, is that by giving the solver less statements to check in the precondition of the action is expected a better performance.

D. Domain 4

In domain 4 a different approach is used for the predicates. Instead of defining the office for each predicate, a general "at" predicate is used with the object and office as parameters. As a result, it is necessary to define all the types of objects in the problem.

The predicates in the domain are as follows:

- **at(what ,office):** object "what" is located at office "office".
- **is_robot(what):** States that "what" is a robot.
- **is_dirty(what):** States that "what" is dirty.
- **is_box (what):** States that "what" is a box.
- **Is_empty(what):** States that "what" is empty.
- **Adjacent (o1,o2):** offices o1 and o2 are horizontally or vertically adjacent. It is assumed that adjacent(o1,o2) = adjacent(o2,o1).

The possible actions and the respective explanations are:

- **Clean-office(where, dirt, robot, empty)**: the robot cleans office "where".
- **Move(what, from, to)**: the robot moves from office "from" to office "to".
- **Push(robot, box, from, to, empty)**: the robot pushes box "box" from office "from" to office "to".

To solve the problem, the following objects are included:

- **oi:** The office, where "i" is the office index. The number of offices goes from 1 to 9.
- **bi:** The boxes, where "i" is the box index. The number of boxes is between 0 and 8.
- **Empty:** An empty condition.
- **Dirt:** A dirty condition.

- **Robot:** A robot.

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

The advantage of this implementation is that it increases the generalization and makes it easier to extend and alter the code. The disadvantage is that the actions have more parameters and might give more unnecessary alternatives to the solver. However, through these proposed alternatives if better ones are given the optimality might rise.

3. PDDL IMPLEMENTATION

In the PDDL implementation the Domain is static, however, the problem changes accordingly with the initial and final states. For simplicity, for each domain, the problem considered for exemplification in the subsequent points is the one proposed in the assignment (problem 4 in table 1). However, all the problems in the study will be analyzed afterwards in the report and the respective code is available in appendix B.

4. TESTING CASES

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

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

	Initial State			**Final State**		
Problem 1						
		Robot			Robot	
Problem 2	🟧	Robot	🟧			
					Robot	
	🟧					
Problem 3	🟧		🟧	Box		
	🟧	Robot				Robot
	🟧 Box	🟧				
Problem 4*	🟧	🟧	🟧			
	🟧 Robot	🟧 Box A	🟧 Box B			Box B
	🟧	🟧	🟧		Box A	Robot
Problem 5	🟧 Box A	🟧 Box B	🟧 Box C	Box F	Box G	Box D
	🟧 Box D	🟧 Box E	🟧 Box F	Box H	Box A	Box B
	🟧 Box G	🟧 Box H	🟧 Robot	Box E	Box C	Robot

* The initial and final states proposed in the exercise.

Colored squares represent dirty offices, while non-colored squares are cleaned offices.

It is important to denote that problem 1 is only to demonstrate that the robot has the full information regarding the world state. This way, no cleaning needs to be implemented, so there are no actions performed by the robot. In Problem 5, it is considered that all the offices are dirty and there are 8 boxes. In order to achieve the maximum complexity, the final state of each box is not adjacent to its initial state. Besides that, the final states of all the boxes correspond to initial states of a box in the state space.

5. RESULT AND ANALYSIS

The performance results for each domain of the five problems considered are presented in the tables below, numerated from 2 to 5. Moreover, the steps taken by the robot are presented in appendix A.

The 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	0	10	22	32	129
Domain 2	0	10	22	30	175
Domain 3	0	12	20	28	199
Domain 4	0	12	20	28	No plan found in time

*Domain 4 problem 5 was timed out before a solution was reached.

According to the values presented in table 2, from a general point of view, it is observable that the number of steps needed to reach the final state increases along with the problem complexity. As an extreme case, problem 1, has null complexity, because no actions were needed during the process.

To each problem, the difference in the number of steps between domains is not relevant, but for the last problem. Precisely, in problems 2, the least complex one, and 5, the most complex one, seems that the simplest implementations, domains 1 and 2, need less steps to conclude the task; whereas for intermediate complexity, problems 3 and 4, domains 3 and 4 take less actions to finish. As a last note in the table's analysis, domain 4, even though it presents a reliable performance among the others, for the maximum complexity (problem 5) a plan cannot be found in a time limit of 10 seconds. This output might be originated by the fact that the approach applied is more general and, therefore, more combinations were being considered at each step.

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	1	39	136	4397	31942
Domain 2	1	39	136	4450	125085
Domain 3	1	52	130	212	115922
Domain 4	1	67	130	258	No plan found in time

*Domain 4 problem 5 was timed out before a solution was reached.

In table 3, according to what was mentioned in the previous paragraph, the increment of complexity implies a growth rate in the number of nodes generated, because the number of steps needed to perform a certain task are superior. Again, the extreme case of problem 1 shows that was the first node generated corresponding to the state space is equal to the final solution, no more nodes are needed.

Comparing the values in each problem, the difference on the number of nodes is more marked. Especially, in problems 4 and 5, where the complexities are elevated, the variance between domains can be dramatic. For example, in problem 4, while domain 3 can reach a solution by generating only 212 nodes, more general domains, as 1 and 2, need respectively 4185 and 4238 nodes. In problem 5, there is an inversion on the previous pattern, because domain 1 presents the most efficient solution by a large scale.

According to these values, it is inferable that with more complexity on the problems, the differences between the domains stand out. Besides that, from the last 2 performance analysis, there is not an overall best model in every situation. So, the select always comes with a cost associated, which needs to be considered in parallel with the main purpose of the task.

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	0	16	59	947	9662
Domain 2	0	16	59	961	37631
Domain 3	0	29	47	94	34951
Domain 4	0	34	46	117	No plan found in time

*Domain 4 problem 5 was timed out before a solution was reached.

The number of nodes expanded in table 4 gives an understanding of the nodes examined that were considered as the most valid at each level of the tree. As can be seen by the table values, the relations between domains in the same problem and between problems are the same as in table 3. However, even though the differences are, as expected, slower, the analysis is transversal.

Furthermore, it is important to denote that the differences registered on more complex problems need to have more importance in the final considerations. In the less complex applications, even though there is a difference, as the values are low, the distinct performances do not present extremely relevant distinctions in the output optimality. On the other hand, in the more complex ones, the extreme variance can produce perceptible differences when the robot is acting in a realistic world space.

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

Table 5 – Time consumption in seconds.

	Problem 1	Problem 2	Problem 3	Problem 4	Problem 5
Domain 1	0.210	0.245	0.35	0.315	1.4
Domain 2	0.222	0.307	0.243	0.373	4,3
Domain 3	0.211	0.263	0.314	0.314	4.1
Domain 4	0.291	0.244	0.371	0.415	No plan found in time

*Domain 4 problem 5 was timed out before a solution was reached.

In the last table, the time consumption is the metric to be evaluated. From a global perspective, 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 and expanded. Additionally, even though there are differences between the domains, from problems 1 to 4, the values achieved are satisfactory.

Besides that, even though in problem 5 the time is not desirable, in fact, for the complexity of the implementation, it can be considered acceptable.

Particularly, problem 5 when domain 4 is applied a solution is not reached under the 10 seconds' margin. Taking that into account, in terms of time, domain 4 is not a workable solution. In fact, in more complex problems, such as 3 and 4, this approach always sets the worst results. Furthermore, the values presented in lower complex situations are good, but do not stand out. This way, in terms of time, domain 4 does not present enough advantages.

As a last note on this analysis, it is important to note that CPU resources have a relevant impact during the test phase, due to the low values indicated.

To converge the different analysis, the previous explanations are completed and connected in the following text.

As a goal of finding the best implementation, the domains need to be separated into 2 groups, the generalized code, with domains 3 and 4, and the problem-specific ones, constituted by domains 1 and 2.

In the first group, it is understandable that domain 4 is not a valid solution. Firstly, rarely it outperforms domain 3 and, even more relevant, for extreme complex situations, like problem 5, the time needed to reach the final state is not feasible. Furthermore, it is comprehended that the generalization of an implementation can be beneficial, however, that does not mean that the more generalized code will be the most preeminent to implement.

In the second group, usually the results are extremely similar. An important observation to stand out is that, even though domains 1 and 2 are remarkably similar, the number of steps and, consequently, the nodes expanded and generated, is not always the same. In problems 2 and 3, with low complexity, the results obtained for the metrics exposed are the same, whereas in problems 4 and 5 the differences are substantial. The main reason for this algorithmic behavior relies on the search algorithm used to solve distinct problems. Furthermore, even though the changes between codes are meaningless, in fact, different predicates imply distinct goal definitions, with a clear influence on the robots search decisions. Because of this variance, domain 1, overall, has the best performance.

In general, for the most simple and complex problems (2 and 5) domain 1 presented the best outcomes, while, for intermediate complexities (3 and 4), domain 3 outperforms the latter. To finish, it is essential to denote that for this study, the number of steps was the most clarifying and crucial decisive metric to distinguish between the domains.

6. CONCLUSIONS

All the goals initially proposed were successfully achieved except for problem 5 by domain 4 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.

Furthermore, the metrics chosen to evaluate the models demonstrate a one-way relation with the increase or decrease of the complexity. Besides that, it was observable a strong relation between the number of steps and the nodes expanded, stating that the search-tree considered by the robot is a representation of the environment.

It is important to point out that, for this implementation, there is not an overall best model, since distinct ways of representing the world always come with advantages and disadvantages associated, that lead to different routes. This way, the preferable solution with the solver used always depends on the real-world implementation.

For the preferable domains it can be concluded that, if most of the real-world problems are of intermediate complexity, domain 3 would be preferable to the automated robot. However, if the most frequent case is for extremely simple or complex situations, domain 1 is the preferable. Moreover, with more samples, preferably with an honest representation of the most frequent real-world situation, the domain decision is more informed.

To finish, it is concluded that the result from the solver is heavily dependent on the domain, even if the initial position and goal are the same. So, the output is not necessarily the best solution, and it is important to experiment diverse approaches to the domain.

Appendix A: Planer outputs

Problem 2:

domain 1	domain 2	domain 3	domain 4
`0.00100: (move o2 o3)`	`0.00100: (move o2 o3)`	`0.00100: (move o2 o1)`	`0.00100: (move robot o2 o1)`
`0.00200: (clean-office o3)`	`0.00200: (clean-office o3)`	`0.00200: (clean-office o1)`	`0.00200: (clean-office o1 dirt robot empty)`
`0.00300: (move o3 o2)`	`0.00300: (move o3 o2)`	`0.00300: (move o1 o4)`	`0.00300: (move robot o1 o2)`
`0.00400: (move o2 o1)`	`0.00400: (move o2 o1)`	`0.00400: (move o4 o7)`	`0.00400: (move robot o2 o3)`
`0.00500: (clean-office o1)`	`0.00500: (clean-office o1)`	`0.00500: (clean-office o7)`	`0.00500: (clean-office o3 dirt robot empty)`
`0.00600: (move o1 o4)`	`0.00600: (move o1 o4)`	`0.00600: (move o7 o4)`	`0.00600: (move robot o3 o6)`
`0.00700: (move o4 o7)`	`0.00700: (move o4 o7)`	`0.00700: (move o4 o1)`	`0.00700: (move robot o6 o9)`
`0.00800: (clean-office o7)`	`0.00800: (clean-office o7)`	`0.00800: (move o1 o2)`	`0.00800: (move robot o9 o8)`
`0.00900: (move o7 o8)`	`0.00900: (move o7 o8)`	`0.00900: (move o2 o3)`	`0.00900: (move robot o8 o7)`
`0.01000: (move o8 o5)`	`0.01000: (move o8 o5)`	`0.01000: (clean-office o3)`	`0.01000: (clean-office o7 dirt robot empty)`
		`0.01100: (move o3 o2)`	`0.01100: (move robot o7 o8)`
		`0.01200: (move o2 o5)`	`0.01200: (move robot o8 o5)`

Problem 3:

domain 1	domain 2	domain 3	domain 4
0.00100: (move o5 o8)	0.00100: (move o5 o8)	0.00100: (move o5 o4)	0.00100: (move robot o5 o4)
0.00200: (clean-office o8)	0.00200: (clean-office o8)	0.00200: (clean-office o4)	0.00200: (clean-office o4 dirt robot empty)
0.00300: (move o8 o7)	0.00300: (move o8 o7)	0.00300: (move o4 o1)	0.00300: (move robot o4 o1)
0.00400: (move o7 o4)	0.00400: (move o7 o4)	0.00400: (clean-office o1)	0.00400: (clean-office o1 dirt robot empty)
0.00500: (clean-office o4)	0.00500: (clean-office o4)	0.00500: (move o1 o2)	0.00500: (move robot o1 o2)
0.00600: (move o4 o1)	0.00600: (move o4 o1)	0.00600: (move o2 o3)	0.00600: (move robot o2 o3)
0.00700: (clean-office o1)	0.00700: (clean-office o1)	0.00700: (clean-office o3)	0.00700: (clean-office o3 dirt robot empty)
0.00800: (move o1 o2)	0.00800: (move o1 o2)	0.00800: (move o3 o2)	0.00800: (move robot o3 o6)
0.00900: (move o2 o3)	0.00900: (move o2 o3)	0.00900: (move o2 o5)	0.00900: (move robot o6 o9)
0.01000: (clean-office o3)	0.01000: (clean-office o3)	0.01000: (move o5 o8)	0.01000: (move robot o9 o8)
0.01100: (move o3 o6)	0.01100: (move o3 o6)	0.01100: (clean-office o8)	0.01100: (clean-office o8 dirt robot empty)
0.01200: (move o6 o9)	0.01200: (move o6 o9)	0.01200: (move o8 o7)	0.01200: (move robot o8 o7)
0.01300: (move o9 o8)	0.01300: (move o9 o8)	0.01300: (push ba o7 o4)	0.01300: (push robot box_a o7 o4 empty)
0.01400: (move o8 o7)	0.01400: (move o8 o7)	0.01400: (push ba o4 o1)	0.01400: (push robot box_a o4 o1 empty)
0.01500: (push ba o7 o4)	0.01500: (push ba o7 o4)	0.01500: (move o1 o4)	0.01500: (move robot o1 o4)
0.01600: (push ba o4 o1)	0.01600: (push ba o4 o1)	0.01600: (move o4 o7)	0.01600: (move robot o4 o7)
0.01700: (move o1 o4)	0.01700: (move o1 o4)	0.01700: (clean-office o7)	0.01700: (clean-office o7 dirt robot empty)
0.01800: (move o4 o7)	0.01800: (move o4 o7)	0.01800: (move o7 o4)	0.01800: (move robot o7 o8)
0.01900: (clean-office o7)	0.01900: (clean-office o7)	0.01900: (move o4 o5)	0.01900: (move robot o8 o9)
0.02000: (move o7 o8)	0.02000: (move o7 o8)	0.02000: (move o5 o6)	0.02000: (move robot o9 o6)
0.02100: (move o8 o9)	0.02100: (move o8 o9)		
0.02200: (move o9 o6)	0.02200: (move o9 o6)		

Problem 4:

domain 1	domain 2	domain 3	domain 4
0.00100: (clean-office o4)	0.00100: (clean-office o4)	0.00100: (clean-office o4)	0.00100: (clean-office o4 dirt robot empty)
0.00200: (move o4 o5)	0.00200: (move o4 o5)	0.00200: (move o4 o1)	0.00200: (move robot o4 o1)
0.00300: (move o5 o8)	0.00300: (move o5 o8)	0.00300: (clean-office o1)	0.00300: (clean-office o1 dirt robot empty)
0.00400: (clean-office o8)	0.00400: (clean-office o8)	0.00400: (move o1 o2)	0.00400: (move robot o1 o2)
0.00500: (move o8 o5)	0.00500: (move o8 o5)	0.00500: (clean-office o2)	0.00500: (clean-office o2 dirt robot empty)
0.00600: (push ba o5 o8)	0.00600: (push ba o5 o8)	0.00600: (move o2 o3)	0.00600: (move robot o2 o3)
0.00700: (move o8 o9)	0.00700: (move o8 o9)	0.00700: (clean-office o3)	0.00700: (clean-office o3 dirt robot empty)
0.00800: (clean-office o9)	0.00800: (clean-office o9)	0.00800: (move o3 o6)	0.00800: (move robot o3 o6)
0.00900: (move o9 o6)	0.00900: (move o9 o6)	0.00900: (move o6 o9)	0.00900: (move robot o6 o9)
0.01000: (move o6 o5)	0.01000: (move o6 o5)	0.01000: (clean-office o9)	0.01000: (clean-office o9 dirt robot empty)
0.01100: (clean-office o5)	0.01100: (clean-office o5)	0.01100: (move o9 o8)	0.01100: (move robot o9 o8)
0.01200: (move o5 o6)	0.01200: (move o5 o6)	0.01200: (clean-office o8)	0.01200: (clean-office o8 dirt robot empty)
0.01300: (move o6 o3)	0.01300: (move o6 o3)	0.01300: (move o8 o5)	0.01300: (move robot o8 o5)
0.01400: (clean-office o3)	0.01400: (clean-office o3)	0.01400: (push ba o5 o8)	0.01400: (push robot box_a o5 o8 empty)
0.01500: (move o3 o2)	0.01500: (move o3 o2)	0.01500: (move o8 o5)	0.01500: (move robot o8 o5)
0.01600: (clean-office o2)	0.01600: (clean-office o2)	0.01600: (clean-office o5)	0.01600: (clean-office o5 dirt robot empty)
0.01700: (move o2 o1)	0.01700: (move o2 o1)	0.01700: (move o5 o4)	0.01700: (move robot o5 o8)
0.01800: (clean-office o1)	0.01800: (clean-office o1)	0.01800: (move o4 o7)	0.01800: (move robot o8 o7)
0.01900: (move o1 o4)	0.01900: (move o1 o4)	0.01900: (clean-office o7)	0.01900: (clean-office o7 dirt robot empty)
0.02000: (move o4 o7)	0.02000: (move o4 o7)	0.02000: (move o7 o4)	0.02000: (move robot o7 o8)
0.02100: (clean-office o7)	0.02100: (clean-office o7)	0.02100: (move o4 o5)	0.02100: (move robot o8 o5)
0.02200: (move o7 o8)	0.02200: (move o7 o8)	0.02200: (move o5 o6)	0.02200: (move robot o5 o6)
0.02300: (move o8 o9)	0.02300: (move o8 o9)	0.02300: (push bb o6 o9)	0.02300: (push robot box_b o6 o9 empty)
0.02400: (move o9 o6)	0.02400: (move o9 o6)	0.02400: (move o9 o6)	0.02400: (move robot o9 o6)
0.02500: (push bb o6 o5)	0.02500: (push bb o6 o5)	0.02500: (clean-office o6)	0.02500: (clean-office o6 dirt robot empty)
0.02600: (move o5 o6)	0.02600: (move o5 o6)	0.02600: (move o6 o9)	0.02600: (move robot o6 o9)
0.02700: (clean-office o6)	0.02700: (clean-office o6)	0.02700: (push bb o9 o6)	0.02700: (push robot box_b o9 o6 empty)
0.02800: (move o6 o9)	0.02800: (move o6 o5)	0.02800: (move o6 o9)	0.02800: (move robot o6 o9)
0.02900: (move o9 o8)	0.02900: (push bb o5 o6)		
0.03000: (move o8 o5)	0.03000: (move o6 o9)		

`0.03100: (push bb o5 o6)`			
`0.03200: (move o6 o9)`			

Problem 5:

domain 1	domain 2	domain 3
0.00100: (clean-office o9)	0.00100: (clean-office o9)	0.00100: (clean-office o9)
0.00200: (move o9 o8)	0.00200: (move o9 o8)	0.00200: (move o9 o6)
0.00300: (push bh o8 o9)	0.00300: (push bh o8 o9)	0.00300: (push bf o6 o9)
0.00400: (move o9 o8)	0.00400: (move o9 o8)	0.00400: (move o9 o6)
0.00500: (clean-office o8)	0.00500: (clean-office o8)	0.00500: (clean-office o6)
0.00600: (move o8 o5)	0.00600: (move o8 o5)	0.00600: (move o6 o5)
0.00700: (push be o5 o8)	0.00700: (push be o5 o8)	0.00700: (push be o5 o6)
0.00800: (move o8 o5)	0.00800: (move o8 o5)	0.00800: (move o6 o5)
0.00900: (clean-office o5)	0.00900: (clean-office o5)	0.00900: (clean-office o5)
0.01000: (move o5 o2)	0.01000: (move o5 o2)	0.01000: (move o5 o4)
0.01100: (push bb o2 o5)	0.01100: (push bb o2 o5)	0.01100: (push bd o4 o5)
0.01200: (move o5 o2)	0.01200: (move o5 o2)	0.01200: (move o5 o4)
0.01300: (clean-office o2)	0.01300: (clean-office o2)	0.01300: (clean-office o4)
0.01400: (move o2 o1)	0.01400: (move o2 o1)	0.01400: (move o4 o1)
0.01500: (push ba o1 o2)	0.01500: (push ba o1 o2)	0.01500: (push ba o1 o4)
0.01600: (move o2 o5)	0.01600: (move o2 o5)	0.01600: (move o4 o5)
0.01700: (move o5 o4)	0.01700: (move o5 o4)	0.01700: (move o5 o2)
0.01800: (push bd o4 o1)	0.01800: (push bd o4 o1)	0.01800: (push bb o2 o1)
0.01900: (move o1 o4)	0.01900: (move o1 o4)	0.01900: (move o1 o2)
0.02000: (clean-office o4)	0.02000: (clean-office o4)	0.02000: (clean-office o2)
0.02100: (move o4 o5)	0.02100: (move o4 o5)	0.02100: (move o2 o5)
0.02200: (push bb o5 o4)	0.02200: (push bb o5 o4)	0.02200: (push bd o5 o2)
0.02300: (move o4 o5)	0.02300: (move o4 o5)	0.02300: (move o2 o5)
0.02400: (move o5 o2)	0.02400: (move o5 o2)	0.02400: (move o5 o4)
0.02500: (push ba o2 o5)	0.02500: (push ba o2 o5)	0.02500: (push ba o4 o5)
0.02600: (move o5 o6)	0.02600: (move o5 o6)	0.02600: (move o5 o4)
0.02700: (move o6 o3)	0.02700: (move o6 o3)	0.02700: (move o4 o7)
0.02800: (push bc o3 o2)	0.02800: (push bc o3 o2)	0.02800: (push bg o7 o4)
0.02900: (move o2 o5)	0.02900: (move o2 o5)	0.02900: (move o4 o5)
0.03000: (move o5 o6)	0.03000: (move o5 o6)	0.03000: (move o5 o8)
0.03100: (push bf o6 o3)	0.03100: (push bf o6 o3)	0.03100: (push bh o8 o7)
0.03200: (move o3 o6)	0.03200: (move o3 o6)	0.03200: (move o7 o8)
0.03300: (clean-office o6)	0.03300: (clean-office o6)	0.03300: (clean-office o8)
0.03400: (move o6 o3)	0.03400: (move o6 o3)	0.03400: (move o8 o9)
0.03500: (push bf o3 o6)	0.03500: (push bf o3 o6)	0.03500: (push bf o9 o8)
0.03600: (move o6 o5)	0.03600: (move o6 o5)	0.03600: (move o8 o9)
0.03700: (move o5 o2)	0.03700: (move o5 o2)	0.03700: (move o9 o6)
0.03800: (push bc o2 o3)	0.03800: (push bc o2 o3)	0.03800: (push be o6 o9)
0.03900: (move o3 o2)	0.03900: (move o3 o2)	0.03900: (move o9 o6)
0.04000: (move o2 o1)	0.04000: (move o2 o1)	0.04000: (move o6 o3)
0.04100: (push bd o1 o2)	0.04100: (push bd o1 o2)	0.04100: (push bc o3 o6)
0.04200: (move o2 o5)	0.04200: (move o2 o5)	0.04200: (move o6 o5)
0.04300: (move o5 o4)	0.04300: (move o5 o4)	0.04300: (move o5 o2)
0.04400: (push bb o4 o1)	0.04400: (push bb o4 o1)	0.04400: (push bd o2 o3)
0.04500: (move o1 o4)	0.04500: (move o1 o4)	0.04500: (move o3 o2)
0.04600: (move o4 o7)	0.04600: (move o4 o7)	0.04600: (move o2 o1)
0.04700: (push bg o7 o4)	0.04700: (push bg o7 o4)	0.04700: (push bb o1 o2)
0.04800: (move o4 o5)	0.04800: (move o4 o5)	0.04800: (move o2 o5)
0.04900: (move o5 o8)	0.04900: (move o5 o8)	0.04900: (move o5 o4)
0.05000: (push be o8 o7)	0.05000: (push be o8 o7)	0.05000: (push bg o4 o1)
0.05100: (move o7 o4)	0.05100: (move o7 o4)	0.05100: (move o1 o2)
0.05200: (move o4 o5)	0.05200: (move o4 o5)	0.05200: (move o2 o5)

0.05300: (push ba o5 o8)	0.05300: (push ba o5 o8)	0.05300: (move o5 o8)
0.05400: (move o8 o9)	0.05400: (move o8 o9)	0.05400: (move o8 o7)
0.05500: (move o9 o6)	0.05500: (move o9 o6)	0.05500: (push bh o7 o4)
0.05600: (push bf o6 o5)	0.05600: (push bf o6 o5)	0.05600: (move o4 o7)
0.05700: (move o5 o2)	0.05700: (move o5 o2)	0.05700: (clean-office o7)
0.05800: (move o2 o3)	0.05800: (move o2 o3)	0.05800: (move o7 o8)
0.05900: (push bc o3 o6)	0.05900: (push bc o3 o6)	0.05900: (push bf o8 o7)
0.06000: (move o6 o3)	0.06000: (move o6 o3)	0.06000: (move o7 o8)
0.06100: (clean-office o3)	0.06100: (clean-office o3)	0.06100: (move o8 o9)
0.06200: (move o3 o2)	0.06200: (move o3 o2)	0.06200: (push be o9 o8)
0.06300: (push bd o2 o3)	0.06300: (push bd o2 o3)	0.06300: (move o8 o9)
0.06400: (move o3 o2)	0.06400: (move o3 o6)	0.06400: (move o9 o6)
0.06500: (move o2 o1)	0.06500: (move o6 o5)	0.06500: (push bc o6 o9)
0.06600: (push bb o1 o2)	0.06600: (push bf o5 o2)	0.06600: (move o9 o8)
0.06700: (move o2 o5)	0.06700: (move o2 o5)	0.06700: (move o8 o5)
0.06800: (move o5 o4)	0.06800: (move o5 o4)	0.06800: (push ba o5 o6)
0.06900: (push bg o4 o1)	0.06900: (push bg o4 o5)	0.06900: (move o6 o3)
0.07000: (move o1 o4)	0.07000: (move o5 o4)	0.07000: (move o3 o2)
0.07100: (move o4 o5)	0.07100: (move o4 o1)	0.07100: (push bb o2 o5)
0.07200: (push bf o5 o4)	0.07200: (push bb o1 o4)	0.07200: (move o5 o4)
0.07300: (move o4 o5)	0.07300: (move o4 o5)	0.07300: (move o4 o1)
0.07400: (move o5 o6)	0.07400: (move o5 o2)	0.07400: (push bg o1 o2)
0.07500: (push bc o6 o5)	0.07500: (push bf o2 o1)	0.07500: (move o2 o1)
0.07600: (move o5 o8)	0.07600: (move o1 o4)	0.07600: (clean-office o1)
0.07700: (move o8 o9)	0.07700: (move o4 o5)	0.07700: (move o1 o4)
0.07800: (push bh o9 o6)	0.07800: (push bg o5 o2)	0.07800: (push bh o4 o1)
0.07900: (move o6 o5)	0.07900: (move o2 o5)	0.07900: (move o1 o4)
0.08000: (move o5 o8)	0.08000: (move o5 o8)	0.08000: (move o4 o7)
0.08100: (push ba o8 o9)	0.08100: (push ba o8 o5)	0.08100: (push bf o7 o4)
0.08200: (move o9 o8)	0.08200: (move o5 o6)	0.08200: (move o4 o5)
0.08300: (move o8 o5)	0.08300: (move o6 o9)	0.08300: (move o5 o8)
0.08400: (push bc o5 o8)	0.08400: (push bh o9 o8)	0.08400: (push be o8 o7)
0.08500: (move o8 o5)	0.08500: (move o8 o9)	0.08500: (move o7 o8)
0.08600: (move o5 o2)	0.08600: (move o9 o6)	0.08600: (move o8 o9)
0.08700: (push bb o2 o5)	0.08700: (push bc o6 o9)	0.08700: (push bc o9 o8)
0.08800: (move o5 o4)	0.08800: (move o9 o8)	0.08800: (move o8 o5)
0.08900: (move o4 o1)	0.08900: (move o8 o5)	0.08900: (move o5 o6)
0.09000: (push bg o1 o2)	0.09000: (push ba o5 o6)	0.09000: (push ba o6 o9)
0.09100: (move o2 o1)	0.09100: (move o6 o5)	0.09100: (move o9 o6)
0.09200: (clean-office o1)	0.09200: (move o5 o4)	0.09200: (move o6 o3)
0.09300: (move o1 o4)	0.09300: (push bb o4 o5)	0.09300: (push bd o3 o6)
0.09400: (push bf o4 o1)	0.09400: (move o5 o2)	0.09400: (move o6 o3)
0.09500: (move o1 o4)	0.09500: (move o2 o1)	0.09500: (clean-office o3)
0.09600: (move o4 o7)	0.09600: (push bf o1 o4)	0.09600: (move o3 o2)
0.09700: (push be o7 o4)	0.09700: (move o4 o1)	0.09700: (push bg o2 o3)
0.09800: (move o4 o7)	0.09800: (clean-office o1)	0.09800: (move o3 o2)
0.09900: (clean-office o7)	0.09900: (move o1 o4)	0.09900: (move o2 o1)
0.10000: (move o7 o8)	0.10000: (push bf o4 o1)	0.10000: (push bh o1 o2)
0.10100: (push bc o8 o7)	0.10100: (move o1 o4)	0.10100: (move o2 o5)
0.10200: (move o7 o8)	0.10200: (move o4 o7)	0.10200: (move o5 o4)
0.10300: (move o8 o5)	0.10300: (push be o7 o4)	0.10300: (push bf o4 o1)
0.10400: (push bb o5 o8)	0.10400: (move o4 o7)	0.10400: (move o1 o2)
0.10500: (move o8 o9)	0.10500: (clean-office o7)	0.10500: (move o2 o5)
0.10600: (move o9 o6)	0.10600: (move o7 o8)	0.10600: (push bb o5 o4)
0.10700: (push bh o6 o5)	0.10700: (push bh o8 o7)	0.10700: (move o4 o5)

0.10800: (move o5 o8)	0.10800: (move o7 o8)	0.10800: (move o5 o2)
0.10900: (move o8 o9)	0.10900: (move o8 o9)	0.10900: (push bh o2 o5)
0.11000: (push ba o9 o6)	0.11000: (push bc o9 o8)	0.11000: (move o5 o6)
0.11100: (move o6 o9)	0.11100: (move o8 o9)	0.11100: (move o6 o3)
0.11200: (move o9 o8)	0.11200: (move o9 o6)	0.11200: (push bg o3 o2)
0.11300: (push bb o8 o9)	0.11300: (push ba o6 o9)	0.11300: (move o2 o5)
0.11400: (move o9 o8)	0.11400: (move o9 o8)	0.11400: (move o5 o6)
0.11500: (move o8 o7)	0.11500: (move o8 o5)	0.11500: (push bd o6 o3)
0.11600: (push bc o7 o8)	0.11600: (push bb o5 o6)	0.11600: (move o3 o6)
0.11700: (move o8 o5)	0.11700: (move o6 o9)	0.11700: (move o6 o9)
0.11800: (move o5 o4)	0.11800: (move o9 o8)	0.11800: (push ba o9 o6)
0.11900: (push be o4 o7)	0.11900: (move o8 o7)	0.11900: (move o6 o5)
0.12000: (move o7 o8)	0.12000: (move o7 o4)	0.12000: (move o5 o8)
0.12100: (move o8 o5)	0.12100: (push be o4 o5)	0.12100: (push bc o8 o9)
0.12200: (push bh o5 o4)	0.12200: (move o5 o8)	0.12200: (move o9 o6)
0.12300: (move o4 o5)	0.12300: (move o8 o7)	0.12300: (move o6 o5)
0.12400: (move o5 o6)	0.12400: (push bh o7 o4)	0.12400: (push bh o5 o8)
0.12500: (push ba o6 o5)	0.12500: (move o4 o5)	0.12500: (move o8 o7)
0.12600: (move o5 o6)	0.12600: (move o5 o8)	0.12600: (move o7 o4)
0.12700: (move o6 o9)	0.12700: (push bc o8 o7)	0.12700: (push bb o4 o5)
0.12800: (push bb o9 o6)	0.12800: (move o7 o8)	0.12800: (move o5 o8)
0.12900: (move o6 o9)	0.12900: (move o8 o9)	0.12900: (move o8 o7)
	0.13000: (push ba o9 o8)	0.13000: (push be o7 o4)
	0.13100: (move o8 o5)	0.13100: (move o4 o5)
	0.13200: (move o5 o6)	0.13200: (move o5 o8)
	0.13300: (push bb o6 o9)	0.13300: (push bh o8 o7)
	0.13400: (move o9 o8)	0.13400: (move o7 o8)
	0.13500: (move o8 o5)	0.13500: (move o8 o9)
	0.13600: (push be o5 o6)	0.13600: (push bc o9 o8)
	0.13700: (move o6 o9)	0.13700: (move o8 o9)
	0.13800: (move o9 o8)	0.13800: (move o9 o6)
	0.13900: (push ba o8 o5)	0.13900: (push ba o6 o9)
	0.14000: (move o5 o4)	0.14000: (move o9 o8)
	0.14100: (move o4 o7)	0.14100: (move o8 o5)
	0.14200: (push bc o7 o8)	0.14200: (push bb o5 o6)
	0.14300: (move o8 o5)	0.14300: (move o6 o5)
	0.14400: (move o5 o4)	0.14400: (move o5 o4)
	0.14500: (push bh o4 o7)	0.14500: (push be o4 o5)
	0.14600: (move o7 o8)	0.14600: (move o5 o8)
	0.14700: (move o8 o5)	0.14700: (move o8 o7)
	0.14800: (push ba o5 o4)	0.14800: (push bh o7 o4)
	0.14900: (move o4 o5)	0.14900: (move o4 o5)
	0.15000: (move o5 o8)	0.15000: (move o5 o8)
	0.15100: (move o8 o9)	0.15100: (push bc o8 o7)
	0.15200: (move o9 o6)	0.15200: (move o7 o8)
	0.15300: (push be o6 o5)	0.15300: (move o8 o9)
	0.15400: (move o5 o8)	0.15400: (push ba o9 o8)
	0.15500: (move o8 o9)	0.15500: (move o8 o9)
	0.15600: (push bb o9 o6)	0.15600: (move o9 o6)
	0.15700: (move o6 o9)	0.15700: (push bb o6 o9)
	0.15800: (move o9 o8)	0.15800: (move o9 o8)
	0.15900: (push bc o8 o9)	0.15900: (move o8 o5)

	0.16000: (move o9 o8)	0.16000: (push be o5 o6)
	0.16100: (move o8 o5)	0.16100: (move o6 o9)
	0.16200: (push be o5 o8)	0.16200: (move o9 o8)
	0.16300: (move o8 o7)	0.16300: (push ba o8 o5)
	0.16400: (move o7 o4)	0.16400: (move o5 o8)
	0.16500: (push ba o4 o5)	0.16500: (move o8 o7)
	0.16600: (move o5 o8)	0.16600: (push bc o7 o8)
	0.16700: (move o8 o7)	0.16700: (move o8 o5)
	0.16800: (push bh o7 o4)	0.16800: (move o5 o4)
	0.16900: (move o4 o7)	0.16900: (push bh o4 o7)
	0.17000: (move o7 o8)	0.17000: (move o7 o8)
	0.17100: (push be o8 o7)	0.17100: (move o8 o5)
	0.17200: (move o7 o8)	0.17200: (push ba o5 o4)
	0.17300: (move o8 o9)	0.17300: (move o4 o7)
	0.17400: (push bc o9 o8)	0.17400: (move o7 o8)
	0.17500: (move o8 o9)	0.17500: (move o8 o9)
		0.17600: (move o9 o6)
		0.17700: (push be o6 o5)
		0.17800: (move o5 o8)
		0.17900: (move o8 o9)
		0.18000: (push bb o9 o6)
		0.18100: (move o6 o9)
		0.18200: (move o9 o8)
		0.18300: (push bc o8 o9)
		0.18400: (move o9 o8)
		0.18500: (move o8 o5)
		0.18600: (push be o5 o8)
		0.18700: (move o8 o7)
		0.18800: (move o7 o4)
		0.18900: (push ba o4 o5)
		0.19000: (move o5 o8)
		0.19100: (move o8 o7)
		0.19200: (push bh o7 o4)
		0.19300: (move o4 o7)
		0.19400: (move o7 o8)
		0.19500: (push be o8 o7)
		0.19600: (move o7 o8)
		0.19700: (move o8 o9)
		0.19800: (push bc o9 o8)
		0.19900: (move o8 o9)

Appendix B: The code

A. Domain 1 and Problem 4

```
(define (domain Cleaning_domain_1)

  (:requirements :strips :adl)

  (:predicates

    (Robot_location ?o)

    (Box_location ?b ?o)

    (Dirty ?o)

    (Clean ?o)

    (Empty ?o)

    (Adjacent ?o1 ?o2)

  )



  (:action move

    :parameters (?o1 ?o2)

    :precondition (and

      (Robot_location ?o1) (or (Adjacent ?o1 ?o2) (Adjacent ?o2 ?o1) ))

    :effect (and (not (Robot_location ?o1))

      (robot_location ?o2) )

  )



  (:action Clean-office

    :parameters (?o)

    :precondition (and

      (Robot_location ?o) (Empty ?o) )

    :effect (and (not (Dirty ?o))

      (Clean ?o))

  )
  (:action Push

    :parameters (?b ?o1 ?o2)

    :precondition (and

      (Robot_location ?o1) (Box_location ?b ?o1) (Empty ?o2)

      (or (Adjacent ?o1 ?o2) (Adjacent ?o2 ?o1) ))

    :effect (and (not (Robot_location ?o1)) (not (Box_location ?b ?o1)) (not
(Empty ?o2))

      (Robot_location ?o2) (Box_location ?b ?o2) (Empty ?o1))

  )

)
```

Problem

```
;The state world of the problem representing a building including nine offices,
between zero and eight boxes and one robot,

;which is the task performer.

;Particularly, each office is defined by two combinations of states (cleaned or
dirty and empty or not empty).

;The robot can move vertically or horizontally between adjacent offices.

;Furthermore, it can move one box to an adjacent empty office.



(define (problem Cleaning_1-4)

  (:domain Cleaning_domain_1)

  (:objects

    o1 o2 o3 o4 o5 o6 o7 o8 o9 ba bb ;bc bd be bf bg bh

  )

  (:init

    (Robot_location o4)



    ;Define location of each box
```

```
(Box_location ba o5)

(Box_location bb o6)

;(Box_location bc o3)

;(Box_location bd o4)

;(Box_location be o5)

;(Box_location bf o6)

;(Box_location bg o7)

;(Box_location bh o8)


;Define dirty offices

(Dirty o1)

(Dirty o2)

(Dirty o3)

(Dirty o4)

(Dirty o5)

(Dirty o6)

(Dirty o7)

(Dirty o8)

(Dirty o9)



;Define clean offices

;(Clean o1)

;(Clean o2)

;(Clean o3)

;(Clean o4)

;(Clean o5)

;(Clean o6)

;(Clean o7)

;(Clean o8)

;(Clean o9)
```

```
   ;Define offices without any box
   (Empty o1)
   (Empty o2)
   (Empty o3)
   (Empty o4)
   ;(Empty o5)
   ;(Empty o6)
   (Empty o7)
   (Empty o8)
   (Empty o9)


   ;Define which offices that is adjacent.
   (Adjacent o1 o2)
   (Adjacent o1 o4)
   (Adjacent o2 o3)
   (Adjacent o2 o5)
   (Adjacent o4 o5)
   (Adjacent o5 o6)
   (Adjacent o3 o6)
   (Adjacent o4 o7)
   (Adjacent o5 o8)
   (Adjacent o7 o8)
   (Adjacent o6 o9)
   (Adjacent o8 o9)


 )
 (:goal
   (and (Robot_location o9)  (Box_location bb o6) (Box_location ba o8)
    (Clean o1) (Clean o2) (Clean o3) (Clean o4) (Clean o5) (Clean o6) (Clean
o7) (Clean o8) (Clean o9)
    )
  )
)
```

B. Domain 2 and Problem 4

```
(define (domain Cleaning_domain_2)
  (:requirements :strips :adl)
  (:predicates
    (Robot_location ?o)
    (Box_location ?b ?o)
    (Dirty ?o)
    (Empty ?o)
    (Adjacent ?o1 ?o2)
  )


  (:action move
    :parameters (?o1 ?o2)
    :precondition (and
      (Robot_location ?o1) (or (Adjacent ?o1 ?o2) (Adjacent ?o2 ?o1) ))
    :effect (and (not (Robot_location ?o1))
      (robot_location ?o2) )
  )


  (:action Clean-office
    :parameters (?o)
    :precondition (and
      (Robot_location ?o) (Empty ?o) )
    :effect (not (Dirty ?o))
    )



  (:action Push
    :parameters (?b ?o1 ?o2)
    :precondition (and
      (Robot_location ?o1) (Box_location ?b ?o1) (Empty ?o2)
      (or (Adjacent ?o1 ?o2) (Adjacent ?o2 ?o1) ))
    :effect (and (not (Robot_location ?o1)) (not (Box_location ?b ?o1)) (not
(Empty ?o2))
      (Robot_location ?o2) (Box_location ?b ?o2) (Empty ?o1))
  )
)
```

Problem

```
;The state world of the problem representing a building including nine offices,
between zero and eight boxes and one robot,
;which is the task performer.
;Particularly, each office is defined by two combinations of states (cleaned or
dirty and empty or not empty).
;The robot can move vertically or horizontally between adjacent offices.
;Furthermore, it can move one box to an adjacent empty office.


(define (problem Cleaning_2-4)
  (:domain Cleaning_domain_2)
  (:objects
    o1 o2 o3 o4 o5 o6 o7 o8 o9 ba bb ;bc bd be bf bg bh
```

```
)

(:init

  (Robot_location o4)



  ;Define location of each box

  (Box_location ba o5)

  (Box_location bb o6)

  ;(Box_location bc o3)

  ;(Box_location bd o4)

  ;(Box_location be o5)

  ;(Box_location bf o6)

  ;(Box_location bg o7)

  ;(Box_location bh o8)



  ;Define dirty offices

  (Dirty o1)

  (Dirty o2)

  (Dirty o3)

  (Dirty o4)

  (Dirty o5)

  (Dirty o6)

  (Dirty o7)

  (Dirty o8)

  (Dirty o9)



  ;Define offices without any box

  (Empty o1)

  (Empty o2)

  (Empty o3)
```

```
    (Empty o4)

    ;(Empty o5)

    ;(Empty o6)

    (Empty o7)

    (Empty o8)

    (Empty o9)



    ;Define which offices that is adjacent.

    (Adjacent o1 o2)

    (Adjacent o1 o4)

    (Adjacent o2 o3)

    (Adjacent o2 o5)

    (Adjacent o4 o5)

    (Adjacent o5 o6)

    (Adjacent o3 o6)

    (Adjacent o4 o7)

    (Adjacent o5 o8)

    (Adjacent o7 o8)

    (Adjacent o6 o9)

    (Adjacent o8 o9)



  )

  (:goal

    (and (Robot_location o9)  (Box_location bb o6) (Box_location ba o8)

      (not(Dirty o1)) (not(Dirty o2)) (not(Dirty o3)) (not(Dirty o4))
(not(Dirty o5)) (not(Dirty o6)) (not(Dirty o7)) (not(Dirty o8)) (not(Dirty o9))

    )

  )

)
```

C. Domain 3 and Problem 4

```
(define (domain Cleaning_domain_3)
  (:requirements :strips :adl)
  (:predicates
    (Robot_location ?o)
    (Box_location ?b ?o)
    (Dirty ?o)
    (Empty ?o)
    (Adjacent ?o1 ?o2)
  )


  (:action move
    :parameters (?o1 ?o2)
    :precondition (and
      (Robot_location ?o1)  (Adjacent ?o1 ?o2) )
    :effect (and (not (Robot_location ?o1))
      (robot_location ?o2) )
  )


  (:action Clean-office
    :parameters (?o)
    :precondition (and
      (Robot_location ?o) (Empty ?o) )
    :effect (not (Dirty ?o))
    )



  (:action Push
    :parameters (?b ?o1 ?o2)

    :precondition (and

      (Robot_location ?o1) (Box_location ?b ?o1) (Empty ?o2)

      (Adjacent ?o1 ?o2) )

    :effect (and (not (Robot_location ?o1)) (not (Box_location ?b ?o1)) (not
(Empty ?o2))

      (Robot_location ?o2) (Box_location ?b ?o2) (Empty ?o1))

  )

)
```

Problem

```
;The state world of the problem representing a building including nine offices,
between zero and eight boxes and one robot,

;which is the task performer.

;Particularly, each office is defined by two combinations of states (cleaned or
dirty and empty or not empty).

;The robot can move vertically or horizontally between adjacent offices.

;Furthermore, it can move one box to an adjacent empty office.



(define (problem Cleaning_3-4)

  (:domain Cleaning_domain_3)

  (:objects

    o1 o2 o3 o4 o5 o6 o7 o8 o9 ba bb; bc bd be bf bg bh

  )

  (:init

    (Robot_location o4)



    ;Define location of each box

    (Box_location ba o5)

    (Box_location bb o6)

    ;(Box_location bc o3)
```

```
;(Box_location bd o4)

;(Box_location be o5)

;(Box_location bf o6)

;(Box_location bg o7)

;(Box_location bh o8)



;Define dirty offices

(Dirty o1)

(Dirty o2)

(Dirty o3)

(Dirty o4)

(Dirty o5)

(Dirty o6)

(Dirty o7)

(Dirty o8)

(Dirty o9)



;Define offices without any box

(Empty o1)

(Empty o2)

(Empty o3)

(Empty o4)

;(Empty o5)

;(Empty o6)

(Empty o7)

(Empty o8)

(Empty o9)



;Define which offices that is adjacent.
```

```
    (Adjacent o1 o2)
    (Adjacent o2 o1)
    (Adjacent o1 o4)
    (Adjacent o4 o1)
    (Adjacent o2 o3)
    (Adjacent o3 o2)
    (Adjacent o2 o5)
    (Adjacent o5 o2)
    (Adjacent o4 o5)
    (Adjacent o5 o4)
    (Adjacent o5 o6)
    (Adjacent o6 o5)
    (Adjacent o3 o6)
    (Adjacent o6 o3)
    (Adjacent o4 o7)
    (Adjacent o7 o4)
    (Adjacent o5 o8)
    (Adjacent o8 o5)
    (Adjacent o7 o8)
    (Adjacent o8 o7)
    (Adjacent o6 o9)
    (Adjacent o9 o6)
    (Adjacent o8 o9)
    (Adjacent o9 o8)



  )
  (:goal
    (and (Robot_location o9)  (Box_location bb o6) (Box_location ba o8)
      (not(Dirty o1)) (not(Dirty o2)) (not(Dirty o3)) (not(Dirty o4))
(not(Dirty o5)) (not(Dirty o6)) (not(Dirty o7)) (not(Dirty o8)) (not(Dirty o9))
```

```
    )
  )
)
```

D. Domain 4 and Problem 4

```
(define (domain Cleaning_domain_4)
  (:requirements :strips :adl )
  (:predicates


    (Adjacent ?o1 ?o2)
    (at ?what ?office)
    (is_robot ?what)
    (is_dirt ?what)
    (is_box ?what)
    (is_empty ?what)
    )




  (:action move
    :parameters (?what ?from ?to)
    :precondition (and
       (Adjacent ?to ?from) (at ?what ?from) (is_robot ?what)
      )
    :effect (and (not (at ?what ?from))
      (at ?what ?to)) )
  (:action Clean-office

    :parameters (?where ?dirt ?robot ?empty)

    :precondition (and

       (is_dirt ?dirt) (at ?robot ?where) (is_robot ?robot) (at ?empty ?where)
(is_empty ?empty))

    :effect  (not (at ?dirt ?where))

    )




  (:action Push

    :parameters (?robot ?box ?from ?to ?empty)

    :precondition (and

       (is_box ?box) (is_robot ?robot) (at ?robot ?from) (at ?box ?from) (at
?empty ?to) (is_empty ?empty)

       (Adjacent ?from ?to))

    :effect (and (not (at ?robot ?from)) (not (at ?box ?from)) (at ?empty
?from) (not(at ?empty ?to))

      (at ?robot ?to) (at ?box ?to) )

  )

)
```

Problem

```
;The state world of the problem representing a building including nine offices,
between zero and eight boxes and one robot,

;which is the task performer.

;Particularly, each office is defined by two combinations of states (cleaned or
dirty and empty or not empty).

;The robot can move vertically or horizontally between adjacent offices.

;Furthermore, it can move one box to an adjacent empty office.



(define (problem Cleaning_4-4)
  (:domain Cleaning_domain_4)
  (:objects
    o1 o2 o3 o4 o5 o6 o7 o8 o9
    box_a box_b empty;box_c box_d box_e box_f box_g box_h
    robot dirt
  )
  (:init
    ;Define the "type" of objects
    (is_robot robot)
    (is_dirt dirt)
    (is_empty empty)


    (is_box box_a)
    (is_box box_b)
    ;(is_box box_c)
    ;(is_box box_d)
    ;(is_box box_e)
    ;(is_box box_f)
    ;(is_box box_g)
    ;(is_box box_h)

    ;Define the position of the objects
    (at robot o4)


    (at box_a o5)
    (at box_b o6)
    ;(at box_c o3)
    ;(at box_d o4)
    ;(at box_e o5)
    ;(at box_f o6)
```

```
;(at box_g o7)

;(at box_h o8)



(at dirt o1)

(at dirt o2)

(at dirt o3)

(at dirt o4)

(at dirt o5)

(at dirt o6)

(at dirt o7)

(at dirt o8)

(at dirt o9)



(at empty o1)

(at empty o2)

(at empty o3)

(at empty o4)

;(at empty o5)

;(at empty o6)

(at empty o7)

(at empty o8)

(at empty o9)



;Define which offices that is adjacent.

(Adjacent o1 o2)

(Adjacent o2 o1)

(Adjacent o1 o4)

(Adjacent o4 o1)

(Adjacent o2 o3)
```

```
    (Adjacent o3 o2)

    (Adjacent o2 o5)

    (Adjacent o5 o2)

    (Adjacent o4 o5)

    (Adjacent o5 o4)

    (Adjacent o5 o6)

    (Adjacent o6 o5)

    (Adjacent o3 o6)

    (Adjacent o6 o3)

    (Adjacent o4 o7)

    (Adjacent o7 o4)

    (Adjacent o5 o8)

    (Adjacent o8 o5)

    (Adjacent o7 o8)

    (Adjacent o8 o7)

    (Adjacent o6 o9)

    (Adjacent o9 o6)

    (Adjacent o8 o9)

    (Adjacent o9 o8)



  )

  (:goal

    (and (at robot o9) (at box_a o8) (at box_b o6)

    (not (at dirt o1)) (not (at dirt o2)) (not (at dirt o3)) (not (at dirt o4))

    (not (at dirt o5)) (not (at dirt o6)) (not (at dirt o7)) (not (at dirt o8))
(not (at dirt o9))

    )

  )

)
```