Practical Exercise 1: Planner

The Coffee Server

Planning and Approximate Reasoning

Universitat Rovira i Virgili

Master in Artificial Inteligence

1^{st} Semester

Authors
Guillermo Bernárdez
Juan Lao

November 6, 2016

Abstract

The STRIPS algorithm, an automated planner developed by Richard Fikes and Nils Nilsson in 1971 at SRI International, can be applied to any state-based problem represented by a set of operators and predicates. In this paper we use STRIPS to solve a problem where a robot has to prepare and serve coffee to several offices in a building, designing specific heuristic functions for this domain.

Index of contents

1 Introduction

In this paper, we present a solution for the practical proposed exercise [1] in PAR course [2]. In the following sections our team presents a definition of the problem and analyzes its characteristics, in order to formalize it as a set of operators and predicates to be able to solve it using the STRIPS algorithm [3]. Later our team proposes several heuristic methods to optimize the result of the solver for this specific problem and presents a solution implemented in Java [4]. Finally, a table of results is presented and analyzed.

Appendix A contains the system requirements and the necessary instructions to execute the proposed solution.

2 Definition of the Problem

The problem, called The Coffee Server [1], considers the following scenario:

"There is a squared building composed by 36 offices, which are located in a matrix of 6 rows and 6 columns. From each office it is possible to move (horizontally or vertically) to the adjacent offices. The building has some coffee machines in some offices that can make 1, 2 or 3 cups of coffee at one time.

The people working at the offices may ask for coffee and a robot called "Clooney" is in charge of serving the coffees required. Each office may ask for 1, 2 or 3 coffees but not more. The petitions of coffee are done all at early morning (just when work starts) so that the robot can plan the service procedure. Each petition has to be served in a single service.

The goal is to serve all the drinks to all the offices in an efficient way (minimizing the travel inside the building, in order to not disturb the people working).

The robot will start with a given initial configuration (different for each test), with some petitions of coffee on a subset of the offices in the building. The initial configuration also establishes the positions of the coffee machines and the initial position of the robot. In the goal state no more petitions are pending to be served."

In this practical exercise our team have to design and implement (in Java) a linear planner with a stack of goals (STRIPS) that can discover how to go efficiently from an initial state of the world to the goal state explained before.

3 Analysis of the Problem

This section formalizes the problem as a set of predicates, parameters, operators and constraints, as it is defined in the problem definition [1].

The predicates to be considered are the following:

- `Robot-location(o)`: the robot is in office o.
- `Robot-free`: the robot has no cup of coffee
- `Robot-loaded(n)`: the robot has n cups of coffee
- `Petition(o,n)`: office o wants n cups of coffee
- `Served(o)`: office o has been served, no more coffee is needed in this office
- `Machine(o,n)`: there is a coffee machine in office o that produces n cups of coffee each time (with n equal to 1, 2 or 3)
- `Steps(x)`: the total distance travelled by the robot (calculated with Manhattan distance)

The operators to be considered are the following:

- `Make(o,n)`: the robot makes n cups of coffee in the machine located at office o
  - Preconditions: `Robot-location(o)`, `Robot-free`, `Machine(o,n)`
  - Postconditions: `Robot-loaded(n)`, ¬`Robot-free`
- `Move(o1,o2)`: the robot moves from o1 to o2
  - Preconditions: `Robot-location(o1)`, `Steps(x)` [1].
  - Postconditions: `Robot-location(o2)`,`Steps(x+distance(o1,o2))`, ¬`Robot-location(o1)`, ¬`Steps(x)`
- `Serve(o,n)`: the robot delivers n cups of coffee to office o
  - Preconditions: `Robot-location(o)`, `Robot-loaded(n)`, `Petition(o,n)`
  - Postconditions: `Served(o)`, `Robot-free`, ¬`Petition(o,n)`, ¬`Robot-loaded(n)`

There are three types of parameters to be considered:

- o: this parameter represents a location in the grid in terms of the number of the office. Since we have a finite 6×6 grid, the allowed value for this parameter is oi, where i is a natural number between 1 and 36 (inclusive).
- n: represents the number of coffees. We only consider petitions and machines up to 3 coffees, so it can only take the values 1, 2 and 3.

[1] As we will see in section 5, the `Steps(x)` precondition has been removed from the final implementation, since it always exists in the current state.

- x: this parameter represents a certain number of steps (i.e. a single or accumulated Manhattan distance), and its value can take any natural number (including 0).

Constraints that simplify the problem, as stated in [1]:

1. The robot only makes coffee for a single petition each time. That is, the robot cannot make 3 cups of coffee to serve two different offices. First, will make coffee of one office and serve it, and after will go to the same or another coffee machine to serve the second office.
2. If a petition is of n cups, the robot will make coffee only with a unique machine of capacity n. That is, the robot cannot make coffee with 2 or more machines and accumulate the cups to serve a unique petition.

Without taking into account the `Steps(x)` predicate, the search space of this problem is gigantic: Robot-location has 36 possible parameters, the robot has 4 possible states (free or loaded with 1, 2 or 3 coffees) and each petition can be served or still pending. Taking into account the `Steps(x)` predicate, the search space of this problem becomes infinite, since x can take any value in $\mathbb{N}$.

"STRIPS uses a GPS-like means-end analysis strategy [5]. This combination of means-ends analysis and formal theorem-proving methods allows objects (world models) much more complex and general than any of those used in GPS and provides more powerful search heuristics than those found in theorem-proving programs" [3]. Thanks to this property, we can solve our problem although the search space is infinite.

Finally, we want to note the following special situation: when there exist a petition for a number of coffees that no machine is able to make. In this case, it is impossible to achieve the goal state from the initial one, since the initial and goal states actually belong to different subgraphs of the search space that doesn't share any connection

4 Planning Algorithm

This section presents and analyzes the different heuristics proposed for the solution of the problem, explaining the domain knowledge behind them. We recall from the notes of the course [6] that the STRIPS algorithm has three potential situations in which a problem-dependent strategy may be deployed in order to improve the performance of the planner (or even to make it work correctly):

1. When choosing an operator to achieve a certain predicate.
2. When sorting a set of predicates to add them to the stack.
3. When instantiating a parameter of a partially instantiated predicate.

We study each case separately.

4.1 Choosing an operator

When the top element of the stack is a predicate that is not contained in the current state, the STRIPS algorithm looks for an operator that adds this predicate to the current state. In a general

example, it could be the case that there were several operators that had that predicate in their postcondition list, some of them perhaps representing a better choice than others. In that case, after studying the specific problem, we may be able to design an intelligent strategy to choose the most appropriate operator.

In the Coffee Server problem, however, each predicate satisfies that only appears in the post-condition list of one of the operators, so we don't have to deal with this problem.

4.2 Sorting a Set of Predicates

In general, given a set of predicates, the order in which we add its predicates to the stack can become crucial in order to make the planner more efficient, or even to avoid loops. When dealing with a specific problem, a previous analysis of it might suggest us some strategies about how to sort the predicates appropriately.

In the Coffee Server problem we distinguish two kinds of set of predicates depending on the sorting process: firstly, the one associated with the goal state, which is the only that can contain `Served(o)` predicates; and secondly, the sets of predicates related to the precondition lists of the three operators. We detail the domain-knowledge applied to each of them separately,

Sorting the Goal State

A reasonable goal state consists in a set of one or more `Served(o)` predicates plus an optional single `Robot-location(o)` to state the final office position of the robot. A goal state can potentially contain other predicates, but they are not relevant for this heuristic.

First of all, it is straightforward to see that all `Served(o)` predicates should be placed in the stack on top of the `Robot-location(o)`. Then, a more complex strategy is required to sort the different `Served(o)` predicates. We point out that the order of these predicates may highly determine the number of steps of the itinerary followed by the robot, which is just the quantity that we would like to minimize.

We propose an heuristic function based on sorting the `Served(o)` predicates by looking for sequences of nearby ones (considering the Manhattan distance) starting from the initial and, it it is stated in the goal state, from the final position as well. More specifically:

1. We start looking for the `Served(o)` predicate whose position parameter is the closest to the initial position of the robot; it will be the first element taken into account by the planning process. This step defines the top part of the stack.

2. In the next step, if the goal state defines a final position for the robot and there are still unordered `Served(o)` predicates, we look for the `Served(o)` predicate whose position parameter is the closest to the final position of the robot; it will become the last `Served(o)` element taken into account by the planner. This step defines the bottom part of the stack

3. Until all `Served(o)` predicates are ordered, we repeat:

   (a) We consider the last `Served(o)` predicate added to the top part of the stack, and look for its closest Served(o9 predicate among the remaining ones; it will be the following `Served(o)` predicate in that top part of the stack.

(b) If there are still unordered `Served(o)` predicates and the goal state defines a final position, we consider now the last `Served(o)` predicate added to the bottom part of the stack, and again look for its closest `Served(o)` predicate among the remaining ones; it will become the top `Served(o)` predicate in that bottom part of the stack.

This is a fairly simple process that allows us to obtain a reasonable order of serving the commands, avoiding considering two consecutive far petitions while there are closer ones, as well as taking into account the final state (provided it is stated in the goal state) besides the initial one so that, if possible, the robot does not serve the last coffee request far from its final position.

Note that our heuristic does not guarantee an optimal organization at all; it does not even take into account the locations of the coffee machines when sorting the `Served(o)` predicates. A more sophisticated heuristic may lead to solutions closer (or equal) to the optimal one, but in that case it is actually resolving the entire planning problem beforehand.

Sorting the precondition list of an operator

When adding the predicates of a precondition list, we realize that:

- If the precondition list corresponds to the `Move(o1,o2)` operator, it doesn't matter the order in which its predicates are added to the stack.
- If the precondition list corresponds to the `Make([o],n)` operator, it is desirable to add its predicates in the following order
  - Firstly, `Robot-location([o])`;
  - Then `Machine([o], n)`;
  - Lastly, `Robot-free`.

  In that way, the top element of the stack is `Robot-free` (to ensure that the robot is not holding coffees already), the second one `Machine([o], n)` (to look for the location of a Machine that can make a given number of coffees), and finally we validate `Robot-location([o])` (to make the robot be at the location of the selected machine).
- If the precondition list corresponds to the `Serve(o,[n])` operator, there is also a preferred order when adding its predicates to the stack:
  - Firstly, `Robot-location(o)`;
  - Then `Robot-loaded([n])`;
  - Lastly, `Petition(o,[n])`.

  Thus, the first predicate to be evaluated is `Petition(o,[n])` (to look for the number of coffees requested at that office), the next one is `Robot-loaded([n])` (to make sure that it holds the requested number of coffees), and `Robot-location(o)` afterwards (to make the robot be at the office where it must serve those coffees).
- We point out that, when sorting the precondition list of a `Make(o,n)` or `Serve(o,n)` operator, different orders than the ones shown above may lead to fool behaviours of the robot, or even to stop the planner without finding a solution; these effects are consequences of illogical or nonexisting matches for the required instantiations.

On top of the stack	On the middle	At the bottom	No matters
Petition Robot-free	Robot-loaded Machine	Robot-location	Steps

Table 1: Preferences that must satisfy a general order.

We also note that the all orders stated above are compatible with a general order provided that it respects the priorities shown in Table 1, where it doesn't matter the relative order between predicates that share the same priority.

For instance, given a set of predicates that comes from a precondition list, one valid order for looking for the predicates (and add them to the stack if found) is the following:

`Steps(x)`, `Robot-location(o)`, `Machine(o,n)`, `Robot-loaded(n)`, `Robot-free`, `Petition(o,n)`

4.3 Instantiating a Parameter

When considering predicates that are partially instantiated, the STRIPS algorithm searches for possible values of the non-instantiated parameters taking into account the current state at that moment. In a specific problem, when different candidates for a given instantiation are found, the domain-knowledge of that problem can help to define an heuristic function in order to find the most appropriate value among the candidates.

In the Coffee Server problem, if the sorting of the sets of predicates is made as we have explained in the previous subsection, the STRIPS algorithm can only find the following partially instantiated predicates[2]: Machine([o],n) (coming from the precondition list of the Make([o],n) operator); Petition(o,[n])(from Serve(o,[n]) operator); Robot-location([o1]) (from Move([o1],o2) operator); Note that the parameters between brackets are the uninstantiated ones.

For each of these predicates, the STRIPS algorithm tries to obtain a valid value for its parameter by matching the corresponding predicate with predicates in the current state that have the same name and, if the predicate we want to instantiate has already an instantiated parameter, share the same value of this parameter as well. In our case, whereas the algorithm only returns a single match for Petition(o,[n]) and Robot-location([o1]), it can actually return a set of several valid matches for Machine([o],n).

When we have different candidates for the position of the Machine predicate, as we are interested in minimizing the number of steps, our strategy is to choose the position o' among them that minimizes

$$d_{Manhattan}(o_0, o') + d_{Manhattan}(o', o_f)$$

, Where o_0 is the current location of the robot and o_f the location of the petition that is being considered at that specific moment [3].

With this heuristic function we guarantee that the robot always goes to the coffee machine that, given the current location, implies the lower number of steps required to go to make the coffees and then go to serve them.

[2]one may note that steps([x]) is not in the list. This is due to the fact that, in our implementation, the predicate `Steps(x)` is not contained in the precondition list of the `Move(o1,o2)` operator, as it is explained in Section 5

[3]o_f can be found by looking for the position parameter of the unique Serve operator that is present in the stack at that instant.

5 Implementation Design

The implementation of the solution for this problem is divided into two parts:

1. The general STRIPS algorithm.
2. The particular case of the coffee server, as an extension of part 1.

Figure 1 is a simplification of the system class diagram, where classes are grouped by the package in which they are included. A more complete class diagram of the system can be found in Appendix B.

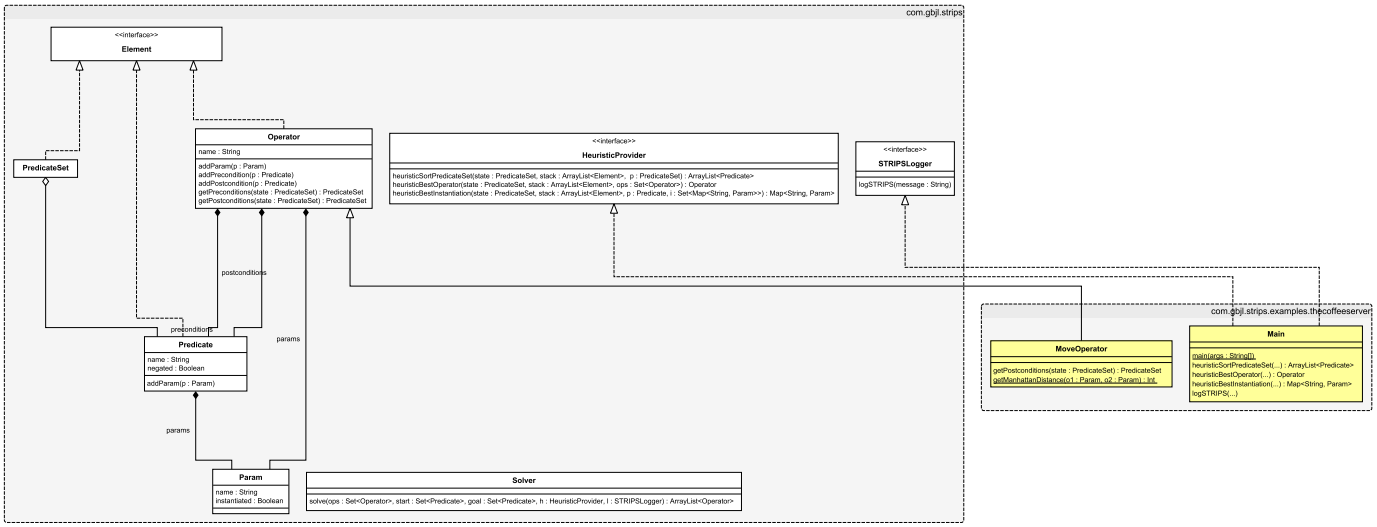


Figure 1: Simple class diagram

Package `com.gbjl.strips` contains the general STRIPS algorithm that, in summary, consists on these classes:

- `Solver`: this class implements the STRIPS algorithm and it is executed when the method `solve(...)` is called with a given set of operators, an initial state, a goal state, a heuristic provider and a logger.
- `HeuristicProvider`: this interface provides the three heuristic functions that the solver uses, as explained in Section 4. Class `Solver` calls those functions when needed.
- `STRIPSLogger`: this interface provides a function for logging messages. Class `Solver` calls this function several times to give information about what it is doing, as well as changes in the current state or the current stack.
- `Element`: this interface represents an element of the stack. In this simplification it has no methods, but internally it provides a function for the instantiation of parameters.
- `Param`: this class represents a parameter of a predicate or operator. It has a name and it can be instantiated (or not).

- `Predicate`: this class represents a predicate. It has a name, it can be negated (or not) and it contains a list of parameters.
- `Operator`: this class represents an operator. It has a name, a list of parameters, a list of preconditions and a list of postconditions. It can be extended with a subclass in order to implement advanced features, such as the `Move` operator of our coffee server problem, which requires a special case for managing the postcondition of `Steps(x+d)`.
- `PredicateSet`: this class represents a set of predicates.

Package `com.gbjl.strips.examples.thecoffeeserver` contains the particular case of the coffee server that consists on these classes:

- `Main`: this class is the start point of the problem. It defines all the required operators (with its parameters, preconditions and postconditions), reads the initial and goal states from a given file and executes the STRIPS algorithm using `com.gbjs.strips.Solve`. It also provides the required heuristic functions and the logger function, that stores the algorithm log in a given file.
- `MoveOperator`: this class extends from `Operator`. It adds the required parameters, preconditions and postconditions to itself and overrides the `getPostconditions(...)` method, to be able to return the predicate `Steps(x+d)` based on the current state `Steps(x)` predicate. In order to simplify the problem, predicate `Steps(x)` has not been included in the list of preconditions of this operator, since it always exists in the current state.

6 Results

In this section we present and compare the results of several testing cases solved by our system:

- Case 1: the example proposed in the problem definition. It consists on 5 petitions and 5 machines, and the proposal solves it with 25 steps.
- Case 2: a random example. It consists on 10 petitions and 10 machines.
- Case 3: another random example, but without a `Robot-location(o)` predicate in the goal state. It consists on 5 petitions and 5 machines.
- Case 4: a difficult case for our heuristic functions, that solve the problem in a very inefficient way. This is an example of a worst-case scenario for our solution: all petitions are distributed on two ends of the grid, and satisfy that all their suitable machines are located in their opposite end. We expect the robot to be constantly crossing without optimizing the travels at all.
- Case 5: an impossible case, where there is a petition of 1 coffees but no machines. It is expected that the program throws an error.
- Case 6: a random case with a lot of petitions (11) but few machines (3).

We point out that the exact input for each of these cases can be fount in Appendix C.

Table 2 shows the performance of our algorithm measured in the number of steps the robot must do in order to achieve the goal state. This number is compared with the optimal solution, found using an algorithm not presented in this paper. Appendix C contains a detailed representation of the state and stack evolution of each execution of these tests.

Case	1 cup petition	2 cup petition	3 cup petition	1 cup machines	2 cup machines	3 cup machines	Final steps	Optimal solution
1	3	1	1	2	2	1	25	25
2	6	2	2	2	2	6	56	52
3	2	2	1	1	1	3	43	39
4	1	1	1	1	1	1	42	26
5	1	0	0	0	0	0	N/A	N/A
6	6	3	2	1	1	1	69	61

Table 2: Results of the different cases

In case 5, the program throws the following message:

```
Error. Predicate "Machine([o],1)" cannot be instantiated in the
   state "Petition(o1,1);Steps(0);Robot-free;Robot-location(o1);"
```

In case 4, as we expected, we obtain the worst result by far. The robot, as we can see with much more detail in Appendix C, Case 4, instead of taking advantage of the crossings from one end to the other to serve the petitions of both sides at the same time, does respect the sequence of nearest petitions that the heuristic provides it and, consequently, delivers the petitions with a larger number of steps than required.

As the table shows, our solution is near the optimal solution for the major part of cases, but it is very inefficient for specific cases where the heuristic function do not sort the `Served(o)` predicates in a good way, since it does not take in account the position of the machines. However, this table is short and does not represents correctly the performance of our solution.

In order to have a better idea of the performance of the algorithm, we also created 300 random cases, where each case has a random number of machines between 1 and 36 and a random number of petitions between 1 and 36. All cases can be solved, since there exists a machine that makes N coffees for each petition of N coffees.

Figures 3, 3 and CHART3 represent the relation between the number of petitions, the number of machines and the number of steps required to solve the problem.

In 3 we can see how the number of petitions is directly correlated with the number of needed steps. Due to the low number of offices in the building, it is hard to determine the kind of correlation between both variables. However, from our point of view, it looks like an exponential function.

In 3 we can see how the number of machines define a top limit to the number of needed steps. This limit is an inverse correlation between the two variables. Under this limit, the number of steps required to solve the problem does not follow any special distribution.

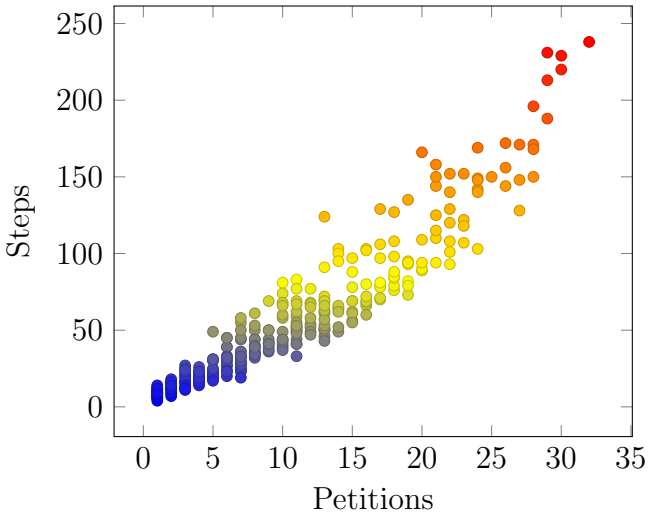


Figure 2: Relation between petitions and steps

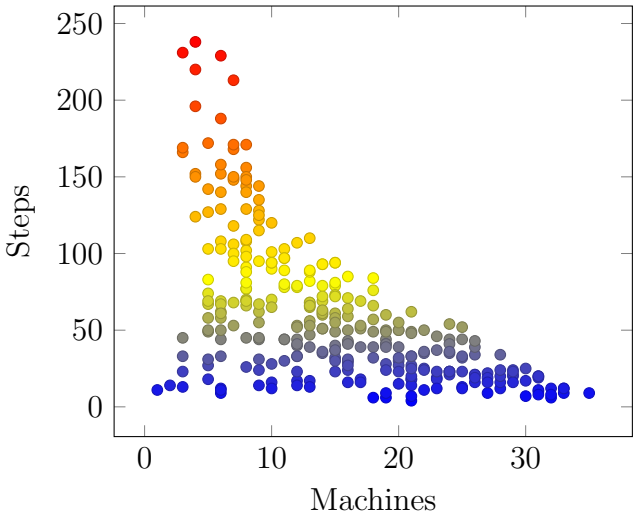


Figure 3: Relation between machines and steps

7 Conclusions and Future Work

In sections 2 and 3, our team presented a definition of the problem and analyzed its characteristics, in order to formalize it as a set of operators and predicates to be able to solve it using the STRIPS algorithm.

Later, in section 4, our team proposed several heuristic functions to optimize the result of the solver for this specific problem.

In section 5, we presented a solution implemented in Java and we explained the details of the object oriented architecture.

Finally, a table of results was presented and analyzed, and we concluded that our solution is near to the optimal one, but in some cases can be very inefficient.

In general, due to the nature of the STRIPS algorithm, this program does not find the optimal plan for serving coffee with the minimum number of robot steps. Nevertheless, it can be close to it

in many situations thanks to the proposed heuristic functions. Optimal solutions for this problem may be found using, for example, a non-linear planning algorithm with regression, but this subject is out of the scope of this paper.

References

[1] PAR course. *MIA-MEISISI: Practical Exercise 1: Planner implementation*. Universitat Rovira i Virgili (URV), Tarragona, Spain, 2016. https://moodle.urv.cat/moodle/pluginfile.php/2303642/mod_resource/content/4/PlannerExercise16-17.pdf

[2] PAR course. *Guia docent 2016_17*. Universitat Rovira i Virgili (URV), Tarragona, Spain, 2016. https://moodle.urv.cat/docnet/guia_docent/index.php?centre=17&ensenyament=1768&assignatura=17685204&font=12&any_academic=2016_17&modalitat=p

[3] Richard E. Fikes and Nils J. Nilsson. *STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving*. Stanford Research Institute, Menlo Park, California, 1971. http://ai.stanford.edu/~nilsson/OnlinePubs-Nils/PublishedPapers/strips.pdf

[4] JAVA. https://www.java.com/

[5] Ernst, G. and Newell, A. *GPS: A Case Study in Generality and Problem Solving*. ACM Monograph Series. Academic Press, New York, New York, 1969.

[6] PAR course. *Lecture 3-3 Algorithm of the linear planner with a stack of objectives*. Universitat Rovira i Virgili (URV), Tarragona, Spain, 2016. https://www.youtube.com/watch?v=8Zt3_EFdtPw

Appendix A: Instructions to Execute the Program

System requirements

- Microsoft Windows Vista or higher. (older versions are not supported by Microsoft[4] at the date of the publication of this paper).
- Java JDK 1.8[5]

Step 1: build

In the command line console, go to the project directory and execute the following command:

```
build-thecoffeeserver.bat
```

Step 2: run

Now, to run the program, execute the following command:

```
java -jar out/thecoffeeserver.jar <input file> <output file>
```

where:

- `<input file>` is a file containing the initial and goal states, with the format stated in the problem definition.
- `<output file>` is the file where the program will store a detailed log of the solving process, including the plan (as a list of operators) to achieve the goal state from the initial state.

To run the input proposed in the problem definition, execute the following command:

```
java -jar out/thecoffeeserver.jar docs/thecoffeeserver/case1.txt
   out/output.txt
```

The output of the program will be stored in out/output.txt.

[4] https://support.microsoft.com/en-us/help/13853/windows-lifecycle-fact-sheet
[5] http://www.oracle.com/technetwork/es/java/javase/downloads/index.html

Appendix B: Complete Class Diagram

This is a complete class diagram of the system. Note that relation multiplicities have been omitted.

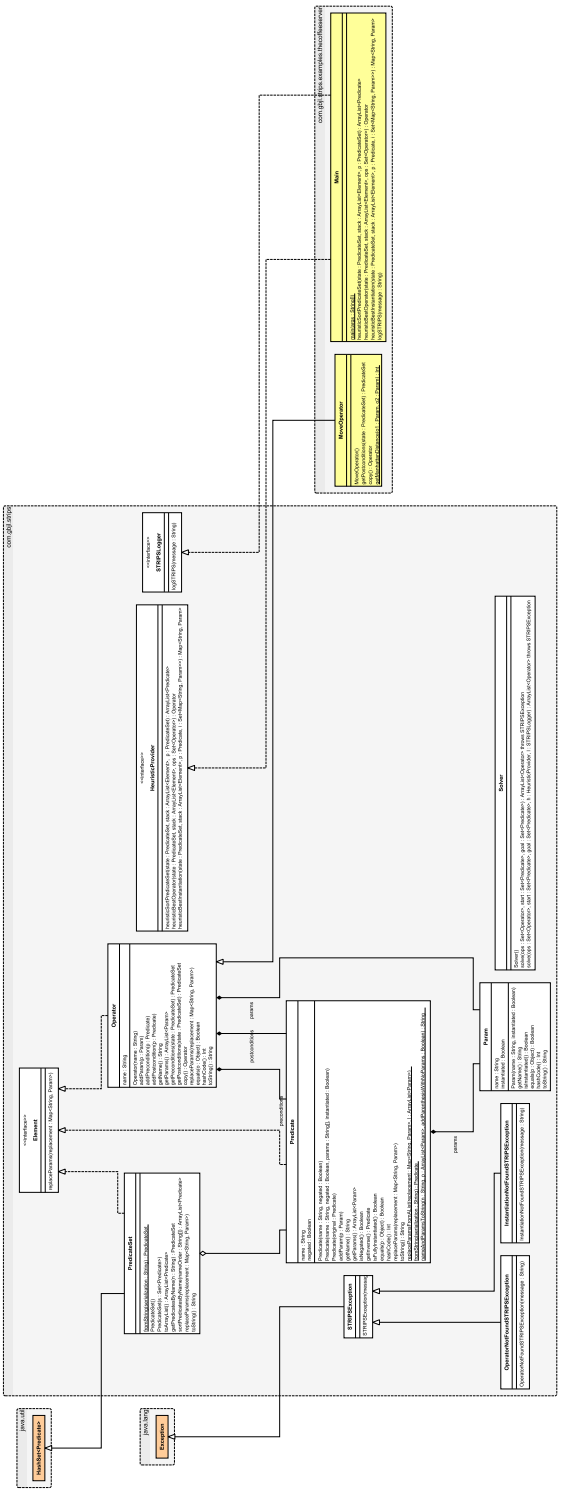


Figure 4: Complete class diagram

Appendix C: Test Results

This appendix contains a detailed representation of the state and stack evolution of each test execution proposed in section REFERENCIASECCIONTESTS.

For reasons of space, we abbreviated some predicates with the following dictionary:

- Robot-loaded → RL
- Robot-free → RF
- Robot-location → R
- Steps → ST
- Machine → M
- Petition → P
- Served → S

For the same reason, when we represent a state using a 6x6 board with predicates in the corresponding cells, we omitted those parameters that reference office locations, since they are implicitly represented. For example, if `M(2)` is written in the cell (2,3) of a board, it means `Machine(o14,2)`.

Case 1

Input File

```
InitialState=Robot-location(o1);Machine(o4,3);Machine(o8,1);Machine
   (o21,2);Machine(o23,1);Machine(o31,2);Petition(o3,1); Petition(
   o11,3);Petition(o12,1);Petition(o13,2);Petition(o25,1);

GoalState=Robot-location(o7);Served(o3);Served(o11);Served(o12);
   Served(o13);Served(o25);
```

State evolution

R		P(1)	M(3)		
	M(1)			P(3)	P(1)
P(2)					
		M(2)		M(1)	
P(1)					
M(2)					

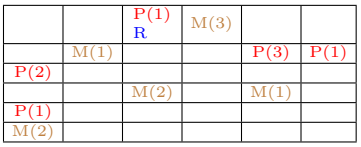
Robot-free; Steps(0)
Initial state

		P(1)	M(3)		
	R M(1)			P(3)	P(1)
P(2)					
		M(2)		M(1)	
P(1)					
M(2)					

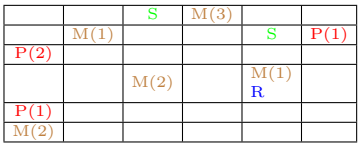
Robot-free; Steps(2)
Intermediate state 1

		P(1)	M(3)		
	R M(1)			P(3)	P(1)
P(2)					
		M(2)		M(1)	
P(1)					
M(2)					

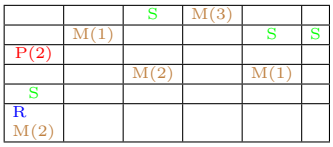
Robot-loaded(1); Steps(2)
Intermediate state 2



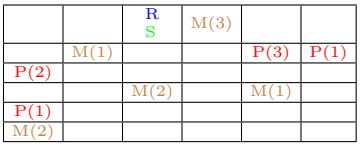
Robot-loaded(1); Steps(4)
Intermediate state 3



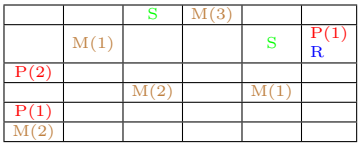
Robot-loaded(1); Steps(9)
Intermediate state 10



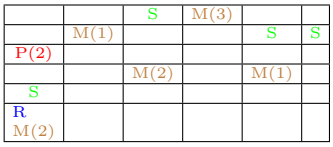
Robot-free; Steps(21)
Intermediate state 17



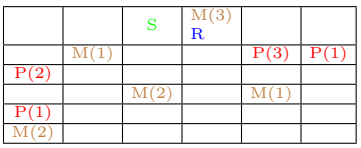
Robot-free; Steps(4)
Intermediate state 4



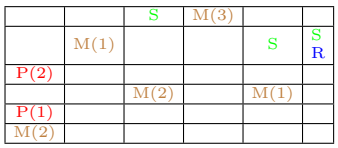
Robot-loaded(1); Steps(12)
Intermediate state 11



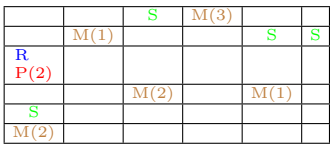
Robot-loaded(2); Steps(21)
Intermediate state 18



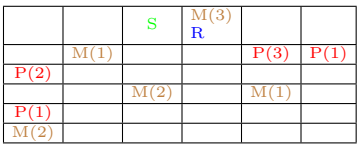
Robot-free; Steps(5)
Intermediate state 5



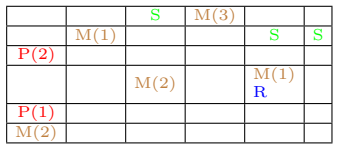
Robot-free; Steps(12)
Intermediate state 12



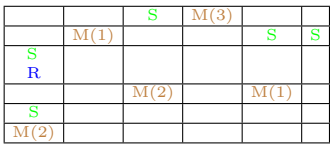
Robot-loaded(2); Steps(24)
Intermediate state 19



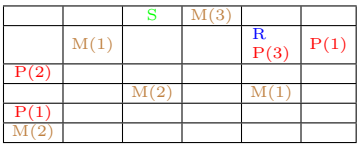
Robot-loaded(3); Steps(5)
Intermediate state 6



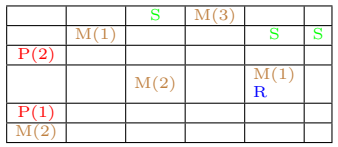
Robot-free; Steps(15)
Intermediate state 13



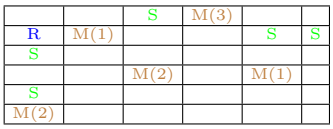
Robot-free; Steps(24)
Intermediate state 20



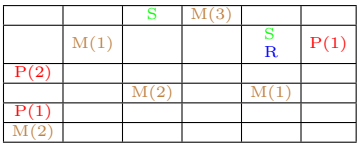
Robot-loaded(3); Steps(7)
Intermediate state 7



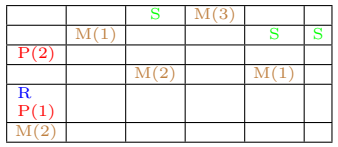
Robot-loaded(1); Steps(15)
Intermediate state 14



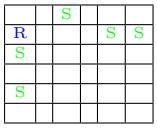
Robot-free; Steps(25)
Final state



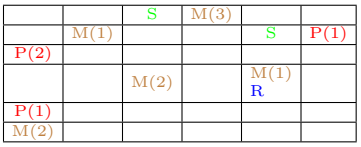
Robot-free; Steps(7)
Intermediate state 8



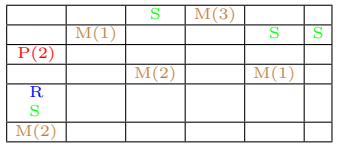
Robot-loaded(1); Steps(20)
Intermediate state 15



Goal state



Robot-free; Steps(9)
Intermediate state 9



Robot-free; Steps(20)
Intermediate state 16

Stack evolution

Served(o3)
Served(o11)
Served(o12)
Served(o25)
Served(o13)
Robot-location(o7)
{S(o3), S(o12), S(o13), S(o25), R(o7), S(o11)}

Stack #1

Petition(o3,[n])
Robot-loaded([n])
Robot-location(o3)
{R(o3), RL([n]), P(o3,[n])}
Serve(o3,[n])
Served(o11)
Served(o12)
Served(o25)
Served(o13)
Robot-location(o7)
{S(o3), S(o12), S(o13), S(o25), R(o7), S(o11)}

Stack #2

Robot-loaded(1)
Robot-location(o3)
{R(o3), RL(1), P(o3,1)}
Serve(o3,1)
Served(o11)
Served(o12)
Served(o25)
Served(o13)
Robot-location(o7)
{S(o3), S(o12), S(o13), S(o25), R(o7), S(o11)}

Stack #3

Robot-free
Machine([o],1)
Robot-location([o])
{M([o],1), R([o]), RF}
Make([o],1)
Robot-location(o3)
{R(o3), RL(1), P(o3,1)}
Serve(o3,1)
Served(o11)
Served(o12)
Served(o25)
Served(o13)
Robot-location(o7)
{S(o3), S(o12), S(o13), S(o25), R(o7), S(o11)}

Stack #4

Machine([o],1)
Robot-location([o])
{M([o],1), R([o]), RF}
Make([o],1)
Robot-location(o3)
{R(o3), RL(1), P(o3,1)}
Serve(o3,1)
Served(o11)
Served(o12)
Served(o25)
Served(o13)
Robot-location(o7)
{S(o3), S(o12), S(o13), S(o25), R(o7), S(o11)}

Stack #5

Robot-location(o8)
{M(o8,1), R(o8), RF}
Make(o8,1)
Robot-location(o3)
{R(o3), RL(1), P(o3,1)}
Serve(o3,1)
Served(o11)
Served(o12)
Served(o25)
Served(o13)
Robot-location(o7)
{S(o3), S(o12), S(o13), S(o25), R(o7), S(o11)}

Stack #6

Robot-location([o1])
{R([o1])}
Move([o1],o8)
{M(o8,1), R(o8), RF}
Make(o8,1)
Robot-location(o3)
{R(o3), RL(1), P(o3,1)}
Serve(o3,1)
Served(o11)
Served(o12)
Served(o25)
Served(o13)
Robot-location(o7)
{S(o3), S(o12), S(o13), S(o25), R(o7), S(o11)}

Stack #7

{R(o1)}
Move(o1,o8)
{M(o8,1), R(o8), RF}
Make(o8,1)
Robot-location(o3)
{R(o3), RL(1), P(o3,1)}
Serve(o3,1)
Served(o11)
Served(o12)
Served(o25)
Served(o13)
Robot-location(o7)
{S(o3), S(o12), S(o13), S(o25), R(o7), S(o11)}

Stack #8

Move(o1,o8)
{M(o8,1), R(o8), RF}
Make(o8,1)
Robot-location(o3)
{R(o3), RL(1), P(o3,1)}
Serve(o3,1)
Served(o11)
Served(o12)
Served(o25)
Served(o13)
Robot-location(o7)
{S(o3), S(o12), S(o13), S(o25), R(o7), S(o11)}

Stack #9

{M(o8,1), R(o8), RF}
Make(o8,1)
Robot-location(o3)
{R(o3), RL(1), P(o3,1)}
Serve(o3,1)
Served(o11)
Served(o12)
Served(o25)
Served(o13)
Robot-location(o7)
{S(o3), S(o12), S(o13), S(o25), R(o7), S(o11)}

Stack #10

Make(o8,1)
Robot-location(o3)
{R(o3), RL(1), P(o3,1)}
Serve(o3,1)
Served(o11)
Served(o12)
Served(o25)
Served(o13)
Robot-location(o7)
{S(o3), S(o12), S(o13), S(o25), R(o7), S(o11)}

Stack #11

Robot-location(o3)
{R(o3), RL(1), P(o3,1)}
Serve(o3,1)
Served(o11)
Served(o12)
Served(o25)
Served(o13)
Robot-location(o7)
{S(o3), S(o12), S(o13), S(o25), R(o7), S(o11)}

Stack #12

Robot-location([o1])
{R([o1])}
Move([o1],o3)
{R(o3), RL(1), P(o3,1)}
Serve(o3,1)
Served(o11)
Served(o12)
Served(o25)
Served(o13)
Robot-location(o7)
{S(o3), S(o12), S(o13), S(o25), R(o7), S(o11)}

Stack #13

{R(o8)}
Move(o8,o3)
{R(o3), RL(1), P(o3,1)}
Serve(o3,1)
Served(o11)
Served(o12)
Served(o25)
Served(o13)
Robot-location(o7)
{S(o3), S(o12), S(o13), S(o25), R(o7), S(o11)}

Stack #14

Move(o8,o3)
{R(o3), RL(1), P(o3,1)}
Serve(o3,1)
Served(o11)
Served(o12)
Served(o25)
Served(o13)
Robot-location(o7)
{S(o3), S(o12), S(o13), S(o25), R(o7), S(o11)}

Stack #15

{R(o3), RL(1), P(o3,1)}
Serve(o3,1)
Served(o11)
Served(o12)
Served(o25)
Served(o13)
Robot-location(o7)
{S(o3), S(o12), S(o13), S(o25), R(o7), S(o11)}

Stack #16

Serve(o3,1)
Served(o11)
Served(o12)
Served(o25)
Served(o13)
Robot-location(o7)
{S(o3), S(o12), S(o13), S(o25), R(o7), S(o11)}

Stack #17

Served(o11)
Served(o12)
Served(o25)
Served(o13)
Robot-location(o7)
{S(o3), S(o12), S(o13), S(o25), R(o7), S(o11)}

Stack #18

Petition(o11,[n])
Robot-loaded([n])
Robot-location(o11)
{R(o11), RL([n]), P(o11,[n])}
Serve(o11,[n])
Served(o12)
Served(o25)
Served(o13)
Robot-location(o7)
{S(o3), S(o12), S(o13), S(o25), R(o7), S(o11)}

Stack #19

Robot-loaded(3)
Robot-location(o11)
{R(o11), RL(3), P(o11,3)}
Serve(o11,3)
Served(o12)
Served(o25)
Served(o13)
Robot-location(o7)
{S(o3), S(o12), S(o13), S(o25), R(o7), S(o11)}

Stack #20

Robot-free
Machine([o],3)
Robot-location([o])
{R([o]), RF, M([o],3)}
Make([o],3)
Robot-location(o11)
{R(o11), RL(3), P(o11,3)}
Serve(o11,3)
Served(o12)
Served(o25)
Served(o13)
Robot-location(o7)
{S(o3), S(o12), S(o13), S(o25), R(o7), S(o11)}

Stack #21

Machine([o],3)
Robot-location([o])
{R([o]), RF, M([o],3)}
Make([o],3)
Robot-location(o11)
{R(o11), RL(3), P(o11,3)}
Serve(o11,3)
Served(o12)
Served(o25)
Served(o13)
Robot-location(o7)
{S(o3), S(o12), S(o13), S(o25), R(o7), S(o11)}

Stack #22

Robot-location(o4)
{R(o4), RF, M(o4,3)}
Make(o4,3)
Robot-location(o11)
{R(o11), RL(3), P(o11,3)}
Serve(o11,3)
Served(o12)
Served(o25)
Served(o13)
Robot-location(o7)
{S(o3), S(o12), S(o13), S(o25), R(o7), S(o11)}

Stack #23

Robot-location([o1])
{R([o1])}
Move([o1],o4)
{R(o4), RF, M(o4,3)}
Make(o4,3)
Robot-location(o11)
{R(o11), RL(3), P(o11,3)}
Serve(o11,3)
Served(o12)
Served(o25)
Served(o13)
Robot-location(o7)
{S(o3), S(o12), S(o13), S(o25), R(o7), S(o11)}

Stack #24

{R(o3)}
Move(o3,o4)
{R(o4), RF, M(o4,3)}
Make(o4,3)
Robot-location(o11)
{R(o11), RL(3), P(o11,3)}
Serve(o11,3)
Served(o12)
Served(o25)
Served(o13)
Robot-location(o7)
{S(o3), S(o12), S(o13), S(o25), R(o7), S(o11)}

Stack #25

Move(o3,o4)
{R(o4), RF, M(o4,3)}
Make(o4,3)
Robot-location(o11)
{R(o11), RL(3), P(o11,3)}
Serve(o11,3)
Served(o12)
Served(o25)
Served(o13)
Robot-location(o7)
{S(o3), S(o12), S(o13), S(o25), R(o7), S(o11)}

Stack #26

{R(o4), RF, M(o4,3)}
Make(o4,3)
Robot-location(o11)
{R(o11), RL(3), P(o11,3)}
Serve(o11,3)
Served(o12)
Served(o25)
Served(o13)
Robot-location(o7)
{S(o3), S(o12), S(o13), S(o25), R(o7), S(o11)}

Stack #27

Make(o4,3)
Robot-location(o11)
{R(o11), RL(3), P(o11,3)}
Serve(o11,3)
Served(o12)
Served(o25)
Served(o13)
Robot-location(o7)
{S(o3), S(o12), S(o13), S(o25), R(o7), S(o11)}

Stack #28

Robot-location(o11)
{R(o11), RL(3), P(o11,3)}
Serve(o11,3)
Served(o12)
Served(o25)
Served(o13)
Robot-location(o7)
{S(o3), S(o12), S(o13), S(o25), R(o7), S(o11)}

Stack #29

Robot-location([o1])
{R([o1])}
Move([o1],o11)
{R(o11), RL(3), P(o11,3)}
Serve(o11,3)
Served(o12)
Served(o25)
Served(o13)
Robot-location(o7)
{S(o3), S(o12), S(o13), S(o25), R(o7), S(o11)}

Stack #30

{R(o4)}
Move(o4,o11)
{R(o11), RL(3), P(o11,3)}
Serve(o11,3)
Served(o12)
Served(o25)
Served(o13)
Robot-location(o7)
{S(o3), S(o12), S(o13), S(o25), R(o7), S(o11)}

Stack #31

Move(o4,o11)
{R(o11), RL(3), P(o11,3)}
Serve(o11,3)
Served(o12)
Served(o25)
Served(o13)
Robot-location(o7)
{S(o3), S(o12), S(o13), S(o25), R(o7), S(o11)}

Stack #32

{R(o11), RL(3), P(o11,3)}
Serve(o11,3)
Served(o12)
Served(o25)
Served(o13)
Robot-location(o7)
{S(o3), S(o12), S(o13), S(o25), R(o7), S(o11)}

Stack #33

Serve(o11,3)
Served(o12)
Served(o25)
Served(o13)
Robot-location(o7)
{S(o3), S(o12), S(o13), S(o25), R(o7), S(o11)}

Stack #34

Served(o12)
Served(o25)
Served(o13)
Robot-location(o7)
{S(o3), S(o12), S(o13), S(o25), R(o7), S(o11)}

Stack #35

Petition(o12,[n])
Robot-loaded([n])
Robot-location(o12)
{R(o12), RL([n]), P(o12,[n])}
Serve(o12,[n])
Served(o25)
Served(o13)
Robot-location(o7)
{S(o3), S(o12), S(o13), S(o25), R(o7), S(o11)}

Stack #36

Robot-loaded(1)
Robot-location(o12)
{R(o12), RL(1), P(o12,1)}
Serve(o12,1)
Served(o25)
Served(o13)
Robot-location(o7)
{S(o3), S(o12), S(o13), S(o25), R(o7), S(o11)}

Stack #37

Robot-free
Machine([o],1)
Robot-location([o])
{M([o],1), R([o]), RF}
Make([o],1)
Robot-location(o12)
{R(o12), RL(1), P(o12,1)}
Serve(o12,1)
Served(o25)
Served(o13)
Robot-location(o7)
{S(o3), S(o12), S(o13), S(o25), R(o7), S(o11)}

Stack #38

Machine([o],1)
Robot-location([o])
{M([o],1), R([o]), RF}
Make([o],1)
Robot-location(o12)
{R(o12), RL(1), P(o12,1)}
Serve(o12,1)
Served(o25)
Served(o13)
Robot-location(o7)
{S(o3), S(o12), S(o13), S(o25), R(o7), S(o11)}

Stack #39

Robot-location(o23)
{M(o23,1), R(o23), RF}
Make(o23,1)
Robot-location(o12)
{R(o12), RL(1), P(o12,1)}
Serve(o12,1)
Served(o25)
Served(o13)
Robot-location(o7)
{S(o3), S(o12), S(o13), S(o25), R(o7), S(o11)}

Stack #40

Robot-location([o1])
{R([o1])}
Move([o1],o23)
{M(o23,1), R(o23), RF}
Make(o23,1)
Robot-location(o12)
{R(o12), RL(1), P(o12,1)}
Serve(o12,1)
Served(o25)
Served(o13)
Robot-location(o7)
{S(o3), S(o12), S(o13), S(o25), R(o7), S(o11)}

Stack #41

{R(o11)}
Move(o11,o23)
{M(o23,1), R(o23), RF}
Make(o23,1)
Robot-location(o12)
{R(o12), RL(1), P(o12,1)}
Serve(o12,1)
Served(o25)
Served(o13)
Robot-location(o7)
{S(o3), S(o12), S(o13), S(o25), R(o7), S(o11)}

Stack #42

Move(o11,o23)
{M(o23,1), R(o23), RF}
Make(o23,1)
Robot-location(o12)
{R(o12), RL(1), P(o12,1)}
Serve(o12,1)
Served(o25)
Served(o13)
Robot-location(o7)
{S(o3), S(o12), S(o13), S(o25), R(o7), S(o11)}

Stack #43

{M(o23,1), R(o23), RF}
Make(o23,1)
Robot-location(o12)
{R(o12), RL(1), P(o12,1)}
Serve(o12,1)
Served(o25)
Served(o13)
Robot-location(o7)
{S(o3), S(o12), S(o13), S(o25), R(o7), S(o11)}

Stack #44

Make(o23,1)
Robot-location(o12)
{R(o12), RL(1), P(o12,1)}
Serve(o12,1)
Served(o25)
Served(o13)
Robot-location(o7)
{S(o3), S(o12), S(o13), S(o25), R(o7), S(o11)}

Stack #45

Robot-location(o12)
{R(o12), RL(1), P(o12,1)}
Serve(o12,1)
Served(o25)
Served(o13)
Robot-location(o7)
{S(o3), S(o12), S(o13), S(o25), R(o7), S(o11)}

Stack #46

Robot-location([o1])
{R([o1])}
Move([o1],o12)
{R(o12), RL(1), P(o12,1)}
Serve(o12,1)
Served(o25)
Served(o13)
Robot-location(o7)
{S(o3), S(o12), S(o13), S(o25), R(o7), S(o11)}

Stack #47

{R(o23)}
Move(o23,o12)
{R(o12), RL(1), P(o12,1)}
Serve(o12,1)
Served(o25)
Served(o13)
Robot-location(o7)
{S(o3), S(o12), S(o13), S(o25), R(o7), S(o11)}

Stack #48

Move(o23,o12)
{R(o12), RL(1), P(o12,1)}
Serve(o12,1)
Served(o25)
Served(o13)
Robot-location(o7)
{S(o3), S(o12), S(o13), S(o25), R(o7), S(o11)}

Stack #49

{R(o12), RL(1), P(o12,1)}
Serve(o12,1)
Served(o25)
Served(o13)
Robot-location(o7)
{S(o3), S(o12), S(o13), S(o25), R(o7), S(o11)}

Stack #50

Serve(o12,1)
Served(o25)
Served(o13)
Robot-location(o7)
{S(o3), S(o12), S(o13), S(o25), R(o7), S(o11)}

Stack #51

Served(o25)
Served(o13)
Robot-location(o7)
{S(o3), S(o12), S(o13), S(o25), R(o7), S(o11)}

Stack #52

Petition(o25,[n])
Robot-loaded([n])
Robot-location(o25)
{R(o25), P(o25,[n]), RL([n])}
Serve(o25,[n])
Served(o13)
Robot-location(o7)
{S(o3), S(o12), S(o13), S(o25), R(o7), S(o11)}

Stack #53

Robot-loaded(1)
Robot-location(o25)
{R(o25), P(o25,1), RL(1)}
Serve(o25,1)
Served(o13)
Robot-location(o7)
{S(o3), S(o12), S(o13), S(o25), R(o7), S(o11)}

Stack #54

Robot-free
Machine([o],1)
Robot-location([o])
{M([o],1), R([o]), RF}
Make([o],1)
Robot-location(o25)
{R(o25), P(o25,1), RL(1)}
Serve(o25,1)
Served(o13)
Robot-location(o7)
{S(o3), S(o12), S(o13), S(o25), R(o7), S(o11)}

Stack #55

Machine([o],1)
Robot-location([o])
{M([o],1), R([o]), RF}
Make([o],1)
Robot-location(o25)
{R(o25), P(o25,1), RL(1)}
Serve(o25,1)
Served(o13)
Robot-location(o7)
{S(o3), S(o12), S(o13), S(o25), R(o7), S(o11)}

Stack #56

Robot-location(o23)
{M(o23,1), R(o23), RF}
Make(o23,1)
Robot-location(o25)
{R(o25), P(o25,1), RL(1)}
Serve(o25,1)
Served(o13)
Robot-location(o7)
{S(o3), S(o12), S(o13), S(o25), R(o7), S(o11)}

Stack #57

Robot-location([o1])
{R([o1])}
Move([o1],o23)
{M(o23,1), R(o23), RF}
Make(o23,1)
Robot-location(o25)
{R(o25), P(o25,1), RL(1)}
Serve(o25,1)
Served(o13)
Robot-location(o7)
{S(o3), S(o12), S(o13), S(o25), R(o7), S(o11)}

Stack #58

{R(o12)}
Move(o12,o23)
{M(o23,1), R(o23), RF}
Make(o23,1)
Robot-location(o25)
{R(o25), P(o25,1), RL(1)}
Serve(o25,1)
Served(o13)
Robot-location(o7)
{S(o3), S(o12), S(o13), S(o25), R(o7), S(o11)}

Stack #59

Move(o12,o23)
{M(o23,1), R(o23), RF}
Make(o23,1)
Robot-location(o25)
{R(o25), P(o25,1), RL(1)}
Serve(o25,1)
Served(o13)
Robot-location(o7)
{S(o3), S(o12), S(o13), S(o25), R(o7), S(o11)}

Stack #60

{M(o23,1), R(o23), RF}
Make(o23,1)
Robot-location(o25)
{R(o25), P(o25,1), RL(1)}
Serve(o25,1)
Served(o13)
Robot-location(o7)
{S(o3), S(o12), S(o13), S(o25), R(o7), S(o11)}

Stack #61

Make(o23,1)
Robot-location(o25)
{R(o25), P(o25,1), RL(1)}
Serve(o25,1)
Served(o13)
Robot-location(o7)
{S(o3), S(o12), S(o13), S(o25), R(o7), S(o11)}

Stack #62

Robot-location(o25)
{R(o25), P(o25,1), RL(1)}
Serve(o25,1)
Served(o13)
Robot-location(o7)
{S(o3), S(o12), S(o13), S(o25), R(o7), S(o11)}

Stack #63

Robot-location([o1])
{R([o1])}
Move([o1],o25)
{R(o25), P(o25,1), RL(1)}
Serve(o25,1)
Served(o13)
Robot-location(o7)
{S(o3), S(o12), S(o13), S(o25), R(o7), S(o11)}

Stack #64

{R(o23)}
Move(o23,o25)
{R(o25), P(o25,1), RL(1)}
Serve(o25,1)
Served(o13)
Robot-location(o7)
{S(o3), S(o12), S(o13), S(o25), R(o7), S(o11)}

Stack #65

Move(o23,o25)
{R(o25), P(o25,1), RL(1)}
Serve(o25,1)
Served(o13)
Robot-location(o7)
{S(o3), S(o12), S(o13), S(o25), R(o7), S(o11)}

Stack #66

{R(o25), P(o25,1), RL(1)}
Serve(o25,1)
Served(o13)
Robot-location(o7)
{S(o3), S(o12), S(o13), S(o25), R(o7), S(o11)}

Stack #67

Serve(o25,1)
Served(o13)
Robot-location(o7)
{S(o3), S(o12), S(o13), S(o25), R(o7), S(o11)}

Stack #68

Served(o13)
Robot-location(o7)
{S(o3), S(o12), S(o13), S(o25), R(o7), S(o11)}

Stack #69

Petition(o13,[n])
Robot-loaded([n])
Robot-location(o13)
{P(o13,[n]), RL([n]), R(o13)}
Serve(o13,[n])
Robot-location(o7)
{S(o3), S(o12), S(o13), S(o25), R(o7), S(o11)}

Stack #70

Robot-loaded(2)
Robot-location(o13)
{P(o13,2), RL(2), R(o13)}
Serve(o13,2)
Robot-location(o7)
{S(o3), S(o12), S(o13), S(o25), R(o7), S(o11)}

Stack #71

Robot-free
Machine([o],2)
Robot-location([o])
{R([o]), RF, M([o],2)}
Make([o],2)
Robot-location(o13)
{P(o13,2), RL(2), R(o13)}
Serve(o13,2)
Robot-location(o7)
{S(o3), S(o12), S(o13), S(o25), R(o7), S(o11)}

Stack #72

Machine([o],2)
Robot-location([o])
{R([o]), RF, M([o],2)}
Make([o],2)
Robot-location(o13)
{P(o13,2), RL(2), R(o13)}
Serve(o13,2)
Robot-location(o7)
{S(o3), S(o12), S(o13), S(o25), R(o7), S(o11)}

Stack #73

Robot-location(o31)
{R(o31), RF, M(o31,2)}
Make(o31,2)
Robot-location(o13)
{P(o13,2), RL(2), R(o13)}
Serve(o13,2)
Robot-location(o7)
{S(o3), S(o12), S(o13), S(o25), R(o7), S(o11)}

Stack #74

Robot-location([o1])
{R([o1])}
Move([o1],o31)
{R(o31), RF, M(o31,2)}
Make(o31,2)
Robot-location(o13)
{P(o13,2), RL(2), R(o13)}
Serve(o13,2)
Robot-location(o7)
{S(o3), S(o12), S(o13), S(o25), R(o7), S(o11)}

Stack #75

{R(o25)}
Move(o25,o31)
{R(o31), RF, M(o31,2)}
Make(o31,2)
Robot-location(o13)
{P(o13,2), RL(2), R(o13)}
Serve(o13,2)
Robot-location(o7)
{S(o3), S(o12), S(o13), S(o25), R(o7), S(o11)}

Stack #76

Move(o25,o31)
{R(o31), RF, M(o31,2)}
Make(o31,2)
Robot-location(o13)
{P(o13,2), RL(2), R(o13)}
Serve(o13,2)
Robot-location(o7)
{S(o3), S(o12), S(o13), S(o25), R(o7), S(o11)}

Stack #77

{R(o31), RF, M(o31,2)}
Make(o31,2)
Robot-location(o13)
{P(o13,2), RL(2), R(o13)}
Serve(o13,2)
Robot-location(o7)
{S(o3), S(o12), S(o13), S(o25), R(o7), S(o11)}

Stack #78

Make(o31,2)
Robot-location(o13)
{P(o13,2), RL(2), R(o13)}
Serve(o13,2)
Robot-location(o7)
{S(o3), S(o12), S(o13), S(o25), R(o7), S(o11)}

Stack #79

Robot-location(o13)
{P(o13,2), RL(2), R(o13)}
Serve(o13,2)
Robot-location(o7)
{S(o3), S(o12), S(o13), S(o25), R(o7), S(o11)}

Stack #80

Robot-location([o1])
{R([o1])}
Move([o1],o13)
{P(o13,2), RL(2), R(o13)}
Serve(o13,2)
Robot-location(o7)
{S(o3), S(o12), S(o13), S(o25), R(o7), S(o11)}

Stack #81

{R(o31)}
Move(o31,o13)
{P(o13,2), RL(2), R(o13)}
Serve(o13,2)
Robot-location(o7)
{S(o3), S(o12), S(o13), S(o25), R(o7), S(o11)}

Stack #82

Move(o31,o13)
{P(o13,2), RL(2), R(o13)}
Serve(o13,2)
Robot-location(o7)
{S(o3), S(o12), S(o13), S(o25), R(o7), S(o11)}

Stack #83

{P(o13,2), RL(2), R(o13)}
Serve(o13,2)
Robot-location(o7)
{S(o3), S(o12), S(o13), S(o25), R(o7), S(o11)}

Stack #84

Serve(o13,2)
Robot-location(o7)
{S(o3), S(o12), S(o13), S(o25), R(o7), S(o11)}

Stack #85

Robot-location(o7)
{S(o3), S(o12), S(o13), S(o25), R(o7), S(o11)}

Stack #86

Robot-location([o1])
{R([o1])}
Move([o1],o7)
{S(o3), S(o12), S(o13), S(o25), R(o7), S(o11)}

Stack #87

{R(o13)}
Move(o13,o7)
{S(o3), S(o12), S(o13), S(o25), R(o7), S(o11)}

Stack #88

Move(o13,o7)
{S(o3), S(o12), S(o13), S(o25), R(o7), S(o11)}

Stack #89

{S(o3), S(o12), S(o13), S(o25), R(o7), S(o11)}

Stack #90

Case 2

Input File

```
InitialState=Petition(o33,2);Robot-location(o16);Machine(o25,3);
   Machine(o24,3);Machine(o29,3);Machine(o28,3);Machine(o7,2);
   Petition(o6,2);Machine(o11,1);Petition(o17,1);Petition(o19,3);
   Petition(o13,3);Machine(o3,3);Petition(o18,1);Petition(o31,1);
   Petition(o1,1);Machine(o16,1);Petition(o27,1);Machine(o5,2);
   Machine(o8,3);Petition(o26,1);

GoalState=Served(o6);Served(o13);Robot-location(o7);Served(o1);
   Served(o33);Served(o19);Served(o18);Served(o31);Served(o27);
   Served(o17);Served(o26);
```

State evolution

P(1)		M(3)		M(2)	P(2)
M(2)	M(3)			M(1)	
P(3)			R M(1)	P(1)	P(1)
P(3)					M(3)
M(3)	P(1)	P(1)	M(3)	M(3)	
P(1)		P(2)			

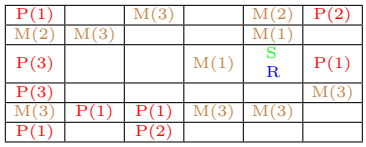
Robot-free; Steps(0)
Initial state

P(1)		M(3)		M(2)	P(2)
M(2)	M(3)			M(1)	
P(3)			R M(1)	P(1)	P(1)
P(3)					M(3)
M(3)	P(1)	P(1)	M(3)	M(3)	
P(1)		P(2)			

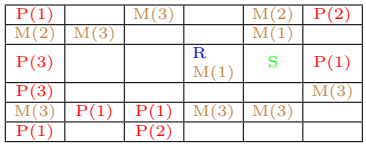
Robot-loaded(1); Steps(0)
Intermediate state 1

P(1)		M(3)		M(2)	P(2)
M(2)	M(3)			M(1)	
P(3)			M(1)	P(1) R	P(1)
P(3)					M(3)
M(3)	P(1)	P(1)	M(3)	M(3)	
P(1)		P(2)			

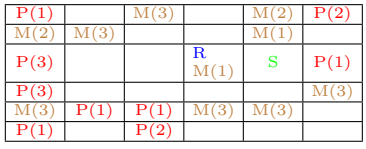
Robot-loaded(1); Steps(1)
Intermediate state 2



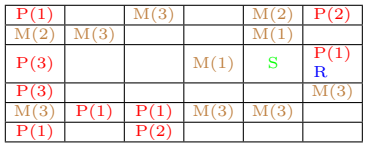
Robot-free; Steps(1)
Intermediate state 3



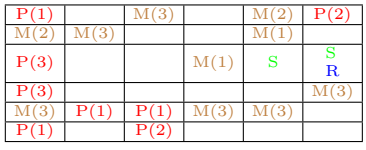
Robot-free; Steps(2)
Intermediate state 4



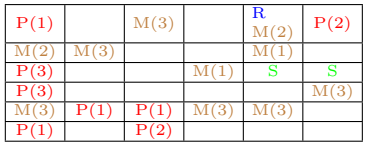
Robot-loaded(1); Steps(2)
Intermediate state 5



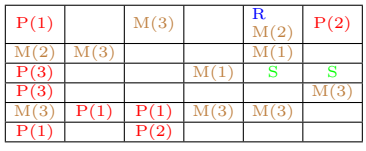
Robot-loaded(1); Steps(4)
Intermediate state 6



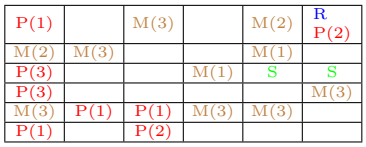
Robot-free; Steps(4)
Intermediate state 7



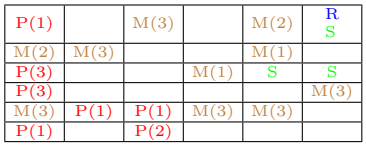
Robot-free; Steps(7)
Intermediate state 8



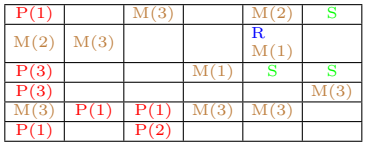
Robot-loaded(2); Steps(7)
Intermediate state 9



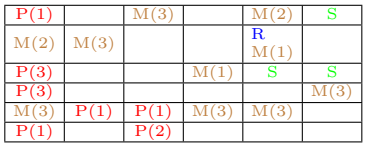
Robot-loaded(2); Steps(8)
Intermediate state 10



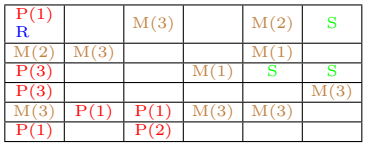
Robot-free; Steps(8)
Intermediate state 11



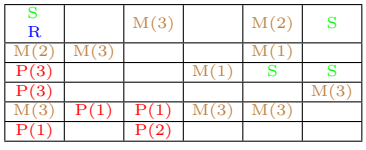
Robot-free; Steps(10)
Intermediate state 12



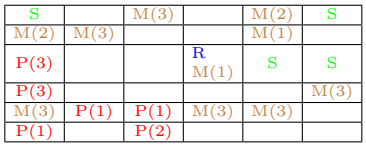
Robot-loaded(1); Steps(10)
Intermediate state 13



Robot-loaded(1); Steps(15)
Intermediate state 14

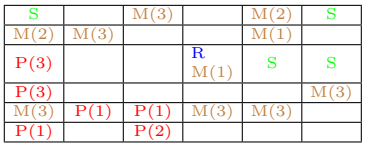


Robot-loaded(1); Steps(15)
Intermediate state 14

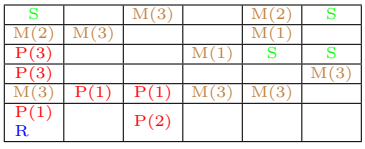


Robot-free; Steps(15)
Intermediate state 15

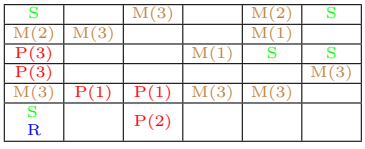
Robot-free; Steps(20)
Intermediate state 16



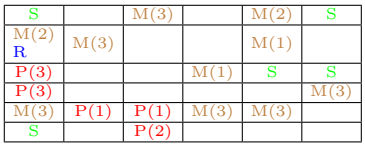
Robot-loaded(1); Steps(20)
Intermediate state 17



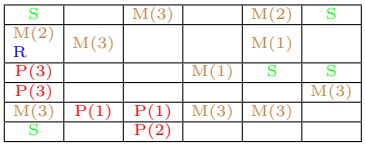
Robot-loaded(1); Steps(26)
Intermediate state 18



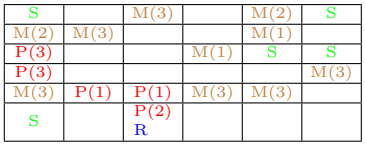
Robot-loaded(1); Steps(26)
Intermediate state 18



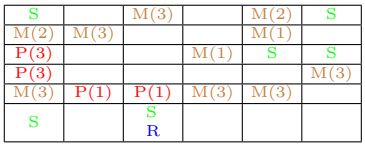
Robot-free; Steps(26)
Intermediate state 19



Robot-free; Steps(30)
Intermediate state 20



Robot-loaded(2); Steps(30)
Intermediate state 21

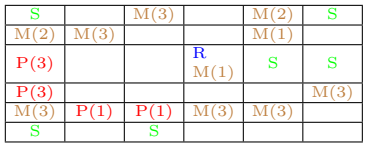


Robot-loaded(2); Steps(36)
Intermediate state 22

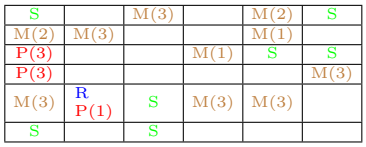
Robot-loaded(2); Steps(36)
Intermediate state 22

Robot-free; Steps(20)
Intermediate state 16

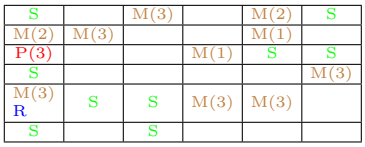
Robot-free; Steps(36)
Intermediate state 23



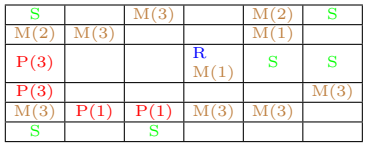
Robot-free; Steps(40)
Intermediate state 24



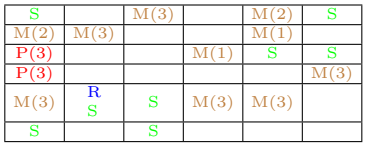
Robot-loaded(1); Steps(50)
Intermediate state 30



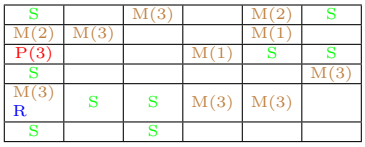
Robot-free; Steps(53)
Intermediate state 36



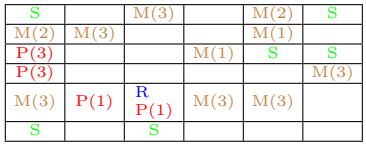
Robot-loaded(1); Steps(40)
Intermediate state 25



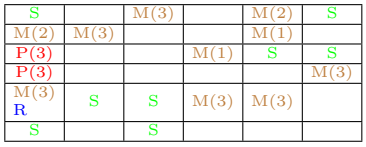
Robot-free; Steps(50)
Intermediate state 31



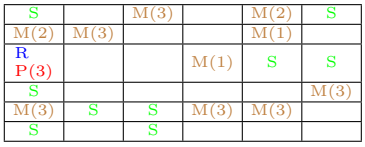
Robot-loaded(3); Steps(53)
Intermediate state 37



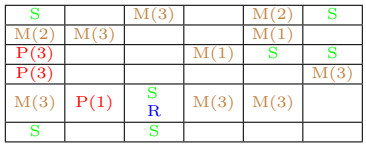
Robot-loaded(1); Steps(43)
Intermediate state 26



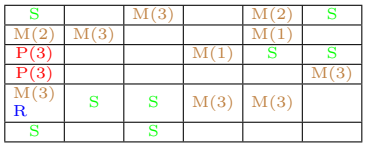
Robot-free; Steps(51)
Intermediate state 32



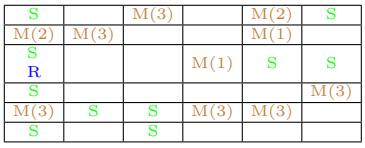
Robot-loaded(3); Steps(55)
Intermediate state 38



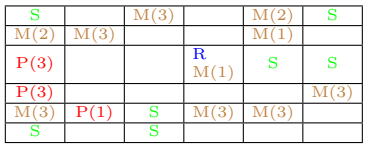
Robot-free; Steps(43)
Intermediate state 27



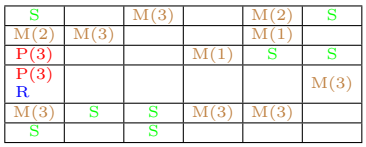
Robot-loaded(3); Steps(51)
Intermediate state 33



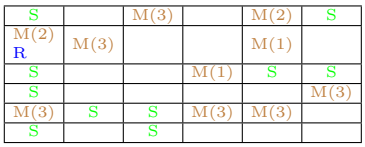
Robot-free; Steps(55)
Intermediate state 39



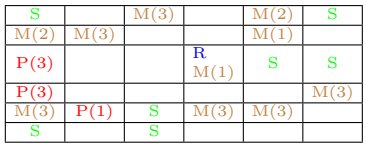
Robot-free; Steps(46)
Intermediate state 28



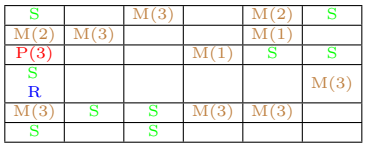
Robot-loaded(3); Steps(52)
Intermediate state 34



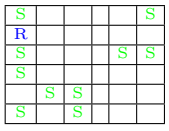
Robot-free; Steps(56)
Final state



Robot-loaded(1); Steps(46)
Intermediate state 29



Robot-free; Steps(52)
Intermediate state 35



Goal state

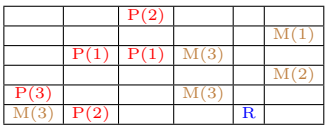
Case 3

Input File

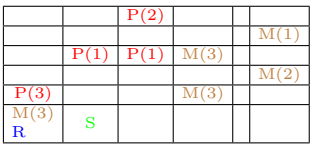
```
InitialState=Machine(o31,3);Petition(o15,1);Robot-location(o35);
   Machine(o12,1);Petition(o3,2);Petition(o25,3);Machine(o24,2);
   Petition(o14,1);Petition(o32,2);Machine(o16,3);Machine(o28,3);
```

```
GoalState=Served(o3);Served(o15);Served(o14);Served(o25);Served(o32
    );
```

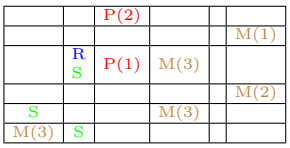
State evolution



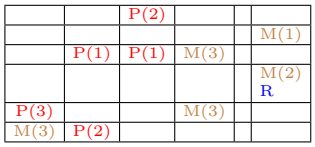
Robot-free; Steps(0)
Initial state



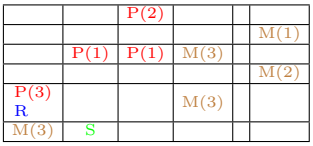
Robot-loaded(3); Steps(10)
Intermediate state 6



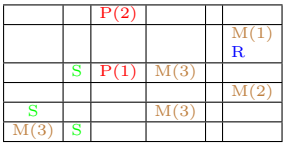
Robot-free; Steps(24)
Intermediate state 12



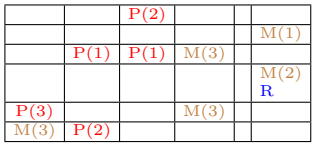
Robot-free; Steps(3)
Intermediate state 1



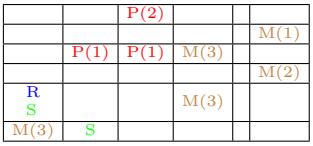
Robot-loaded(3); Steps(11)
Intermediate state 7



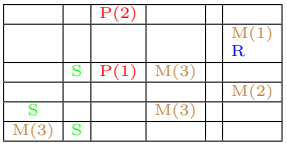
Robot-free; Steps(29)
Intermediate state 13



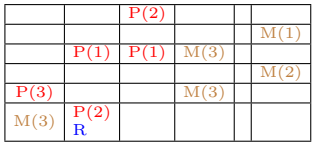
Robot-loaded(2); Steps(3)
Intermediate state 2



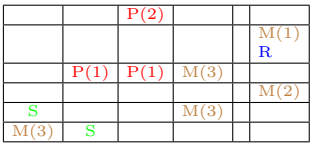
Robot-free; Steps(11)
Intermediate state 8



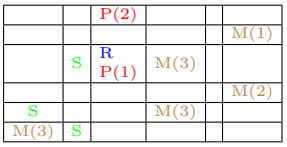
Robot-loaded(1); Steps(29)
Intermediate state 14



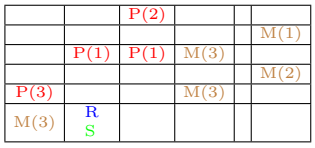
Robot-loaded(2); Steps(9)
Intermediate state 3



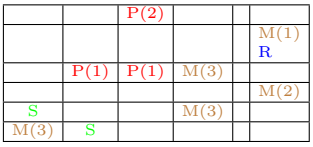
Robot-free; Steps(19)
Intermediate state 9



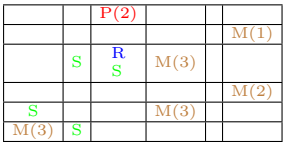
Robot-loaded(1); Steps(33)
Intermediate state 15



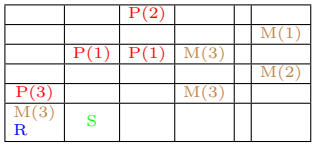
Robot-free; Steps(9)
Intermediate state 4



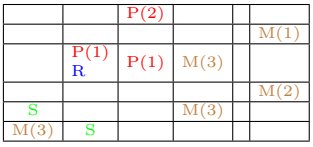
Robot-loaded(1); Steps(19)
Intermediate state 10



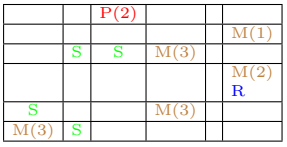
Robot-free; Steps(33)
Intermediate state 16



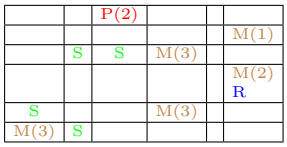
Robot-free; Steps(10)
Intermediate state 5



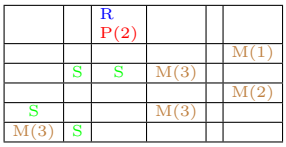
Robot-loaded(1); Steps(24)
Intermediate state 11



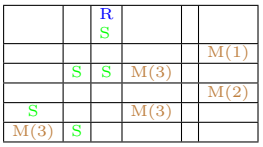
Robot-free; Steps(37)
Intermediate state 17



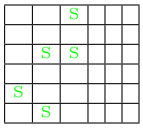
Robot-loaded(2); Steps(37)
Intermediate state 18



Robot-loaded(2); Steps(43)
Intermediate state 19



Robot-free; Steps(43)
Final state



Goal state

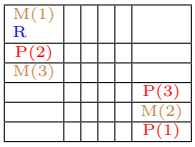
Case 4

Input File

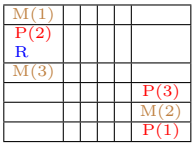
```
InitialState=Robot-location(o1);Machine(o1,1);Petition(o36,1);
   Machine(o30,2);Petition(o7,2);Machine(o13,3);Petition(o24,3);

GoalState=Served(o36);Served(o7);Served(o24);Robot-location(o24);
```

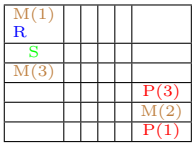
State evolution



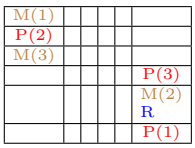
Robot-free; Steps(0)
Initial state



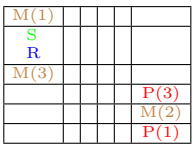
Robot-loaded(2); Steps(17)
Intermediate state 3



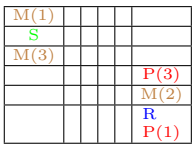
Robot-loaded(1); Steps(18)
Intermediate state 6



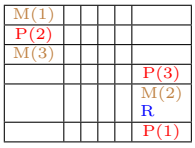
Robot-free; Steps(9)
Intermediate state 1



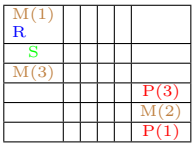
Robot-free; Steps(17)
Intermediate state 4



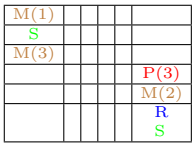
Robot-loaded(1); Steps(28)
Intermediate state 7



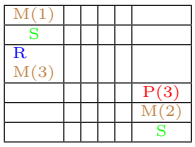
Robot-loaded(2); Steps(9)
Intermediate state 2



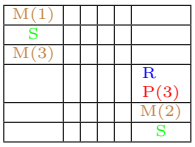
Robot-free; Steps(18)
Intermediate state 5



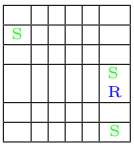
Robot-free; Steps(28)
Intermediate state 8



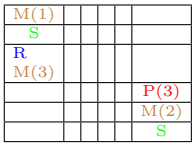
Robot-free; Steps(36)
Intermediate state 9



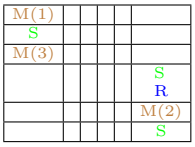
Robot-loaded(3); Steps(42)
Intermediate state 11



Goal state



Robot-loaded(3); Steps(36)
Intermediate state 10



Robot-free; Steps(42)
Final state

Stack evolution

Served(o7)
Served(o36)
Served(o24)
Robot-location(o24)
{S(o7), S(o36), S(o24), R(o24)}

Stack #1

Petition(o7,[n])
Robot-loaded([n])
Robot-location(o7)
{P(o7,[n]), RL([n]), R(o7)}
Serve(o7,[n])
Served(o36)
Served(o24)
Robot-location(o24)
{S(o7), S(o36), S(o24), R(o24)}

Stack #2

Robot-loaded(2)
Robot-location(o7)
{P(o7,2), RL(2), R(o7)}
Serve(o7,2)
Served(o36)
Served(o24)
Robot-location(o24)
{S(o7), S(o36), S(o24), R(o24)}

Stack #3

Robot-free
Machine([o],2)
Robot-location([o])
{R([o]), RF, M([o],2)}
Make([o],2)
Robot-location(o7)
{P(o7,2), RL(2), R(o7)}
Serve(o7,2)
Served(o36)
Served(o24)
Robot-location(o24)
{S(o7), S(o36), S(o24), R(o24)}

Stack #4

Machine([o],2)
Robot-location([o])
{R([o]), RF, M([o],2)}
Make([o],2)
Robot-location(o7)
{P(o7,2), RL(2), R(o7)}
Serve(o7,2)
Served(o36)
Served(o24)
Robot-location(o24)
{S(o7), S(o36), S(o24), R(o24)}

Stack #5

Robot-location(o30)
{R(o30), RF, M(o30,2)}
Make(o30,2)
Robot-location(o7)
{P(o7,2), RL(2), R(o7)}
Serve(o7,2)
Served(o36)
Served(o24)
Robot-location(o24)
{S(o7), S(o36), S(o24), R(o24)}

Stack #6

Robot-location([o1])
{R([o1])}
Move([o1],o30)
{R(o30), RF, M(o30,2)}
Make(o30,2)
Robot-location(o7)
{P(o7,2), RL(2), R(o7)}
Serve(o7,2)
Served(o36)
Served(o24)
Robot-location(o24)
{S(o7), S(o36), S(o24), R(o24)}

Stack #7

{R(o1)}
Move(o1,o30)
{R(o30), RF, M(o30,2)}
Make(o30,2)
Robot-location(o7)
{P(o7,2), RL(2), R(o7)}
Serve(o7,2)
Served(o36)
Served(o24)
Robot-location(o24)
{S(o7), S(o36), S(o24), R(o24)}

Stack #8

Move(o1,o30)
{R(o30), RF, M(o30,2)}
Make(o30,2)
Robot-location(o7)
{P(o7,2), RL(2), R(o7)}
Serve(o7,2)
Served(o36)
Served(o24)
Robot-location(o24)
{S(o7), S(o36), S(o24), R(o24)}

Stack #9

{R(o30), RF, M(o30,2)}
Make(o30,2)
Robot-location(o7)
{P(o7,2), RL(2), R(o7)}
Serve(o7,2)
Served(o36)
Served(o24)
Robot-location(o24)
{S(o7), S(o36), S(o24), R(o24)}

Stack #10

Make(o30,2)
Robot-location(o7)
{P(o7,2), RL(2), R(o7)}
Serve(o7,2)
Served(o36)
Served(o24)
Robot-location(o24)
{S(o7), S(o36), S(o24), R(o24)}

Stack #11

Robot-location(o7)
{P(o7,2), RL(2), R(o7)}
Serve(o7,2)
Served(o36)
Served(o24)
Robot-location(o24)
{S(o7), S(o36), S(o24), R(o24)}

Stack #12

Robot-location([o1])
{R([o1])}
Move([o1],o7)
{P(o7,2), RL(2), R(o7)}
Serve(o7,2)
Served(o36)
Served(o24)
Robot-location(o24)
{S(o7), S(o36), S(o24), R(o24)}

Stack #13

{R(o30)}
Move(o30,o7)
{P(o7,2), RL(2), R(o7)}
Serve(o7,2)
Served(o36)
Served(o24)
Robot-location(o24)
{S(o7), S(o36), S(o24), R(o24)}

Stack #14

Move(o30,o7)
{P(o7,2), RL(2), R(o7)}
Serve(o7,2)
Served(o36)
Served(o24)
Robot-location(o24)
{S(o7), S(o36), S(o24), R(o24)}

Stack #15

{P(o7,2), RL(2), R(o7)}
Serve(o7,2)
Served(o36)
Served(o24)
Robot-location(o24)
{S(o7), S(o36), S(o24), R(o24)}

Stack #16

Serve(o7,2)
Served(o36)
Served(o24)
Robot-location(o24)
{S(o7), S(o36), S(o24), R(o24)}

Stack #17

Served(o36)
Served(o24)
Robot-location(o24)
{S(o7), S(o36), S(o24), R(o24)}

Stack #18

Petition(o36,[n])
Robot-loaded([n])
Robot-location(o36)
{R(o36), P(o36,[n]), RL([n])}
Serve(o36,[n])
Served(o24)
Robot-location(o24)
{S(o7), S(o36), S(o24), R(o24)}

Stack #19

Robot-loaded(1)
Robot-location(o36)
{R(o36), P(o36,1), RL(1)}
Serve(o36,1)
Served(o24)
Robot-location(o24)
{S(o7), S(o36), S(o24), R(o24)}

Stack #20

Robot-free
Machine([o],1)
Robot-location([o])
{M([o],1), R([o]), RF}
Make([o],1)
Robot-location(o36)
{R(o36), P(o36,1), RL(1)}
Serve(o36,1)
Served(o24)
Robot-location(o24)
{S(o7), S(o36), S(o24), R(o24)}

Stack #21

Machine([o],1)
Robot-location([o])
{M([o],1), R([o]), RF}
Make([o],1)
Robot-location(o36)
{R(o36), P(o36,1), RL(1)}
Serve(o36,1)
Served(o24)
Robot-location(o24)
{S(o7), S(o36), S(o24), R(o24)}

Stack #22

Robot-location(o1)
{M(o1,1), R(o1), RF}
Make(o1,1)
Robot-location(o36)
{R(o36), P(o36,1), RL(1)}
Serve(o36,1)
Served(o24)
Robot-location(o24)
{S(o7), S(o36), S(o24), R(o24)}

Stack #23

Robot-location([o1])
{R([o1])}
Move([o1],o1)
{M(o1,1), R(o1), RF}
Make(o1,1)
Robot-location(o36)
{R(o36), P(o36,1), RL(1)}
Serve(o36,1)
Served(o24)
Robot-location(o24)
{S(o7), S(o36), S(o24), R(o24)}

Stack #24

{R(o7)}
Move(o7,o1)
{M(o1,1), R(o1), RF}
Make(o1,1)
Robot-location(o36)
{R(o36), P(o36,1), RL(1)}
Serve(o36,1)
Served(o24)
Robot-location(o24)
{S(o7), S(o36), S(o24), R(o24)}

Stack #25

Move(o7,o1)
{M(o1,1), R(o1), RF}
Make(o1,1)
Robot-location(o36)
{R(o36), P(o36,1), RL(1)}
Serve(o36,1)
Served(o24)
Robot-location(o24)
{S(o7), S(o36), S(o24), R(o24)}

Stack #26

{M(o1,1), R(o1), RF}
Make(o1,1)
Robot-location(o36)
{R(o36), P(o36,1), RL(1)}
Serve(o36,1)
Served(o24)
Robot-location(o24)
{S(o7), S(o36), S(o24), R(o24)}

Stack #27

Make(o1,1)
Robot-location(o36)
{R(o36), P(o36,1), RL(1)}
Serve(o36,1)
Served(o24)
Robot-location(o24)
{S(o7), S(o36), S(o24), R(o24)}

Stack #28

Robot-location(o36)
{R(o36), P(o36,1), RL(1)}
Serve(o36,1)
Served(o24)
Robot-location(o24)
{S(o7), S(o36), S(o24), R(o24)}

Stack #29

Robot-location([o1])
{R([o1])}
Move([o1],o36)
{R(o36), P(o36,1), RL(1)}
Serve(o36,1)
Served(o24)
Robot-location(o24)
{S(o7), S(o36), S(o24), R(o24)}

Stack #30

{R(o1)}
Move(o1,o36)
{R(o36), P(o36,1), RL(1)}
Serve(o36,1)
Served(o24)
Robot-location(o24)
{S(o7), S(o36), S(o24), R(o24)}

Stack #31

Move(o1,o36)
{R(o36), P(o36,1), RL(1)}
Serve(o36,1)
Served(o24)
Robot-location(o24)
{S(o7), S(o36), S(o24), R(o24)}

Stack #32

{R(o36), P(o36,1), RL(1)}
Serve(o36,1)
Served(o24)
Robot-location(o24)
{S(o7), S(o36), S(o24), R(o24)}

Stack #33

Serve(o36,1)
Served(o24)
Robot-location(o24)
{S(o7), S(o36), S(o24), R(o24)}

Stack #34

Served(o24)
Robot-location(o24)
{S(o7), S(o36), S(o24), R(o24)}

Stack #35

Petition(o24,[n])
Robot-loaded([n])
Robot-location(o24)
{P(o24,[n]), RL([n]), R(o24)}
Serve(o24,[n])
Robot-location(o24)
{S(o7), S(o36), S(o24), R(o24)}

Stack #36

Robot-loaded(3)
Robot-location(o24)
{P(o24,3), RL(3), R(o24)}
Serve(o24,3)
Robot-location(o24)
{S(o7), S(o36), S(o24), R(o24)}

Stack #37

Robot-free
Machine([o],3)
Robot-location([o])
{R([o]), RF, M([o],3)}
Make([o],3)
Robot-location(o24)
{P(o24,3), RL(3), R(o24)}
Serve(o24,3)
Robot-location(o24)
{S(o7), S(o36), S(o24), R(o24)}

Stack #38

Machine([o],3)
Robot-location([o])
{R([o]), RF, M([o],3)}
Make([o],3)
Robot-location(o24)
{P(o24,3), RL(3), R(o24)}
Serve(o24,3)
Robot-location(o24)
{S(o7), S(o36), S(o24), R(o24)}

Stack #39

Robot-location(o13)
{R(o13), RF, M(o13,3)}
Make(o13,3)
Robot-location(o24)
{P(o24,3), RL(3), R(o24)}
Serve(o24,3)
Robot-location(o24)
{S(o7), S(o36), S(o24), R(o24)}

Stack #40

Robot-location([o1])
{R([o1])}
Move([o1],o13)
{R(o13), RF, M(o13,3)}
Make(o13,3)
Robot-location(o24)
{P(o24,3), RL(3), R(o24)}
Serve(o24,3)
Robot-location(o24)
{S(o7), S(o36), S(o24), R(o24)}

Stack #41

{R(o36)}
Move(o36,o13)
{R(o13), RF, M(o13,3)}
Make(o13,3)
Robot-location(o24)
{P(o24,3), RL(3), R(o24)}
Serve(o24,3)
Robot-location(o24)
{S(o7), S(o36), S(o24), R(o24)}

Stack #42

Move(o36,o13)
{R(o13), RF, M(o13,3)}
Make(o13,3)
Robot-location(o24)
{P(o24,3), RL(3), R(o24)}
Serve(o24,3)
Robot-location(o24)
{S(o7), S(o36), S(o24), R(o24)}

Stack #43

{R(o13), RF, M(o13,3)}
Make(o13,3)
Robot-location(o24)
{P(o24,3), RL(3), R(o24)}
Serve(o24,3)
Robot-location(o24)
{S(o7), S(o36), S(o24), R(o24)}

Stack #44

Make(o13,3)
Robot-location(o24)
{P(o24,3), RL(3), R(o24)}
Serve(o24,3)
Robot-location(o24)
{S(o7), S(o36), S(o24), R(o24)}

Stack #45

Robot-location(o24)
{P(o24,3), RL(3), R(o24)}
Serve(o24,3)
Robot-location(o24)
{S(o7), S(o36), S(o24), R(o24)}

Stack #46

Robot-location([o1])
{R([o1])}
Move([o1],o24)
{P(o24,3), RL(3), R(o24)}
Serve(o24,3)
Robot-location(o24)
{S(o7), S(o36), S(o24), R(o24)}

Stack #47

{R(o13)}
Move(o13,o24)
{P(o24,3), RL(3), R(o24)}
Serve(o24,3)
Robot-location(o24)
{S(o7), S(o36), S(o24), R(o24)}

Stack #48

Move(o13,o24)
{P(o24,3), RL(3), R(o24)}
Serve(o24,3)
Robot-location(o24)
{S(o7), S(o36), S(o24), R(o24)}

Stack #49

{P(o24,3), RL(3), R(o24)}
Serve(o24,3)
Robot-location(o24)
{S(o7), S(o36), S(o24), R(o24)}

Stack #50

Serve(o24,3)
Robot-location(o24)
{S(o7), S(o36), S(o24), R(o24)}

Stack #51

Robot-location(o24)
{S(o7), S(o36), S(o24), R(o24)}

Stack #52

{S(o7), S(o36), S(o24), R(o24)}

Stack #53

Case 5

Input File

```
InitialState=Robot-location(o1);Petition(o1,1);


GoalState=Robot-location(o1);Served(o1);
```

Result

```
Error. Predicate "Machine([o],1)" cannot be instantiated in the
   state "Petition(o1,1);Steps(0);Robot-free;Robot-location(o1);"
```

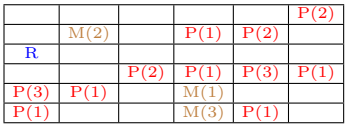
Case 6

Input File

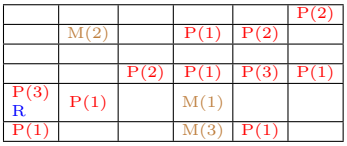
```
InitialState=Petition(o23,3);Petition(o25,3);Machine(o8,2);Machine(
   o34,3);Petition(o35,1);Petition(o21,2);Robot-location(o13);
   Petition(o26,1);Machine(o28,1);Petition(o31,1);Petition(o22,1);
   Petition(o6,2);Petition(o24,1);Petition(o11,2);Petition(o10,1);

GoalState=Served(o11);Served(o22);Served(o23);Served(o24);Served(
   o10);Served(o31);Served(o26);Robot-location(o23);Served(o6);
   Served(o25);Served(o35);Served(o21);
```

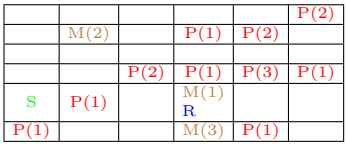
State evolution



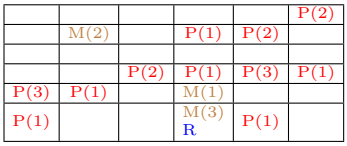
Robot-free; Steps(0)
Initial state



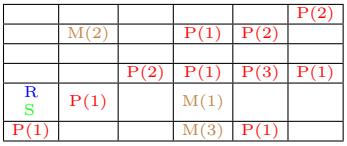
Robot-loaded(3); Steps(10)
Intermediate state 3



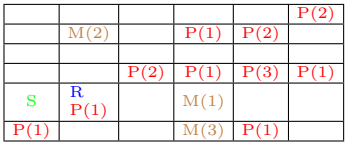
Robot-loaded(1); Steps(13)
Intermediate state 6



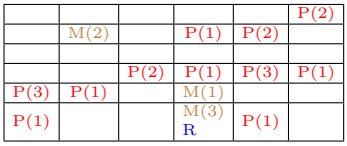
Robot-free; Steps(6)
Intermediate state 1



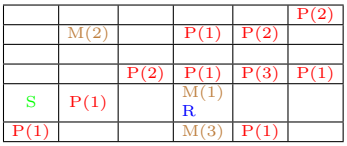
Robot-free; Steps(10)
Intermediate state 4



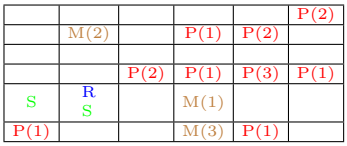
Robot-loaded(1); Steps(15)
Intermediate state 7



Robot-loaded(3); Steps(6)
Intermediate state 2



Robot-free; Steps(13)
Intermediate state 5



Robot-free; Steps(15)
Intermediate state 8

					P(2)
	M(2) R		P(1)	P(2)	
		P(2)	P(1)	P(3)	P(1)
S	S		M(1)		
P(1)			M(3)	P(1)	

Robot-free; Steps(18)
Intermediate state 9

					P(2)
	M(2)		R S	P(2)	
		S	P(1)	P(3)	P(1)
S	S		M(1)		
P(1)			M(3)	P(1)	

Robot-free; Steps(26)
Intermediate state 16

					R P(2)
	M(2)		S	S	
		S	P(1)	P(3)	P(1)
S	S		M(1)		
P(1)			M(3)	P(1)	

Robot-loaded(2); Steps(39)
Intermediate state 23

					P(2)
	M(2) R		P(1)	P(2)	
		P(2)	P(1)	P(3)	P(1)
S	S		M(1)		
P(1)			M(3)	P(1)	

Robot-loaded(2); Steps(18)
Intermediate state 10

					P(2)
	M(2) R		S	P(2)	
		S	P(1)	P(3)	P(1)
S	S		M(1)		
P(1)			M(3)	P(1)	

Robot-free; Steps(28)
Intermediate state 17

					R S
	M(2)		S	S	
		S	P(1)	P(3)	P(1)
S	S		M(1)		
P(1)			M(3)	P(1)	

Robot-free; Steps(39)
Intermediate state 24

					P(2)
	M(2)		P(1)	P(2)	
		R P(2)	P(1)	P(3)	P(1)
S	S		M(1)		
P(1)			M(3)	P(1)	

Robot-loaded(2); Steps(21)
Intermediate state 11

					P(2)
	M(2) R		S	P(2)	
		S	P(1)	P(3)	P(1)
S	S		M(1)		
P(1)			M(3)	P(1)	

Robot-loaded(2); Steps(28)
Intermediate state 18

					S
	M(2)		S	S	
		S	P(1)	P(3)	P(1)
S	S		M(1) R		
P(1)			M(3)	P(1)	

Robot-free; Steps(45)
Intermediate state 25

					P(2)
	M(2)		P(1)	P(2)	
		R S	P(1)	P(3)	P(1)
S	S		M(1)		
P(1)			M(3)	P(1)	

Robot-free; Steps(21)
Intermediate state 12

					P(2)
	M(2)		S	P(2) R	
		S	P(1)	P(3)	P(1)
S	S		M(1)		
P(1)			M(3)	P(1)	

Robot-loaded(2); Steps(31)
Intermediate state 19

					S
	M(2)		S	S	
		S	P(1)	P(3)	P(1)
S	S		M(1) R		
P(1)			M(3)	P(1)	

Robot-loaded(1); Steps(45)
Intermediate state 26

					P(2)
	M(2)		P(1)	P(2)	
		S	P(1)	P(3)	P(1)
S	S		M(1) R		
P(1)			M(3)	P(1)	

Robot-free; Steps(23)
Intermediate state 13

					P(2)
	M(2)		S	S R	
		S	P(1)	P(3)	P(1)
S	S		M(1)		
P(1)			M(3)	P(1)	

Robot-free; Steps(31)
Intermediate state 20

					S
	M(2)		S	S	
		S	P(1)	P(3)	P(1)
S	S		M(1)		
P(1) R			M(3)	P(1)	

Robot-loaded(1); Steps(49)
Intermediate state 27

					P(2)
	M(2)		P(1)	P(2)	
		S	P(1)	P(3)	P(1)
S	S		M(1) R		
P(1)			M(3)	P(1)	

Robot-loaded(1); Steps(23)
Intermediate state 14

					P(2)
	M(2)		S	S	
		S	P(1)	P(3)	P(1)
S	S		M(1)		
P(1)			M(3)	P(1)	

Robot-free; Steps(34)
Intermediate state 21

					S
	M(2)		S	S	
		S	P(1)	P(3)	P(1)
S	S		M(1)		
S R			M(3)	P(1)	

Robot-free; Steps(49)
Intermediate state 28

					P(2)
	M(2)		R P(1)	P(2)	
		S	P(1)	P(3)	P(1)
S	S		M(1)		
P(1)			M(3)	P(1)	

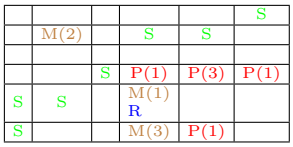
Robot-loaded(1); Steps(26)
Intermediate state 15

					P(2)
	M(2) R		S	S	
		S	P(1)	P(3)	P(1)
S	S		M(1)		
P(1)			M(3)	P(1)	

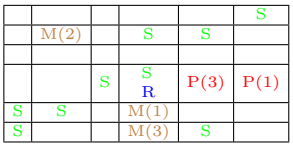
Robot-loaded(2); Steps(34)
Intermediate state 22

					S
	M(2)		S	S	
		S	P(1)	P(3)	P(1)
S	S		M(1) R		
S			M(3)	P(1)	

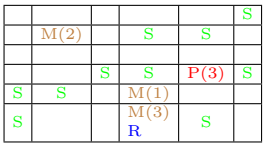
Robot-free; Steps(53)
Intermediate state 29



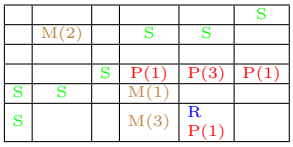
Robot-loaded(1); Steps(53)
Intermediate state 30



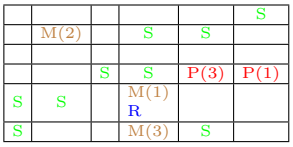
Robot-free; Steps(58)
Intermediate state 36



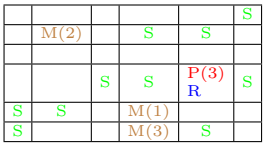
Robot-loaded(3); Steps(66)
Intermediate state 42



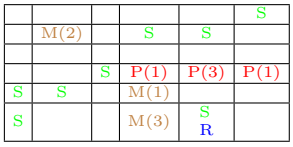
Robot-loaded(1); Steps(55)
Intermediate state 31



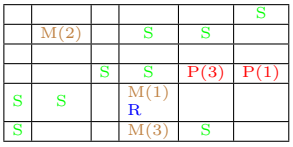
Robot-free; Steps(59)
Intermediate state 37



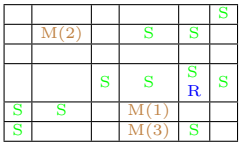
Robot-loaded(3); Steps(69)
Intermediate state 43



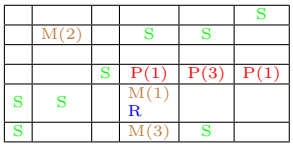
Robot-free; Steps(55)
Intermediate state 32



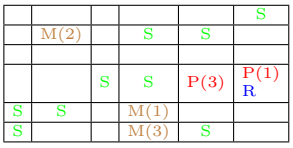
Robot-loaded(1); Steps(59)
Intermediate state 38



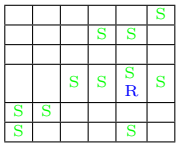
Robot-free; Steps(69)
Final state



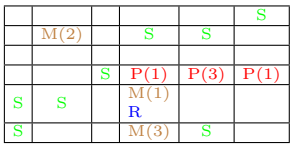
Robot-free; Steps(57)
Intermediate state 33



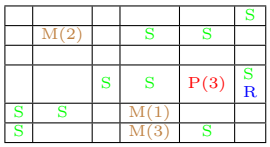
Robot-loaded(1); Steps(62)
Intermediate state 39



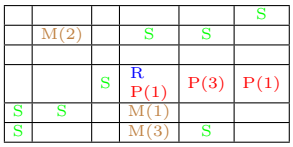
Goal state



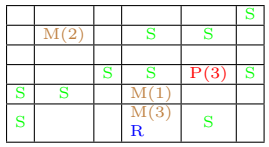
Robot-loaded(1); Steps(57)
Intermediate state 34



Robot-free; Steps(62)
Intermediate state 40



Robot-loaded(1); Steps(58)
Intermediate state 35



Robot-free; Steps(66)
Intermediate state 41