

## **Planning and Approximate Reasoning**

### **Definition of Planner Exercise**

**Group members:**

**SANTIAGO BERNAL IBARRA, MATEUSZ SKIBINSKI**

# **Table of Content**

---

- 1. Introduction to the problem**
- 2. Analysis and formalization of the problem**
- 3. Planning algorithm**
- 4. Implementation design**
- 5. Testing cases and results**
- 6. Instructions to execute the program.**
- 7. Appendix.**

# Introduction

---

In this report we analyze the exercise of blocks world with 2 arms and our implementation of the solution. The block world is a famous planning exercise in artificial intelligence. It consists of stacks of blocks placed on a table and a robot that is going to be in charge of moving the blocks to reorder them, going from an initial state, to a goal state.

The robot has to comply with various restrictions that are predefined for this world, like a heavy block can't go on top of a lighter one and the weight capacity of his left arm is in 1kg, but the right arm can pick up any block. Also, On the table there can be a limited number of stacks of blocks.

We approached this exercise with a design and implementation of a non-linear planner with goal regression using java.

In order to implement the java software we started by defining operators and predicates. Each operator is represented by information that describes the state of the world before and changes to the world after an action is performed, it also has a list of preconditions that have to be met in order for it to be able to perform. Predicates can be described as conditions, for example one of predicates refers to the fact that we can have only two stacks of blocks at any given time, therefore this predicate updates it self after application of some operators. By using predicates we can define different states, and the operators allow us to move from one state to the next. So the basics of the software is to find operators that allows a robot to move blocks from an initial state to a goal state, in the least amount of operators as possible.

# Analysis And Formalization

---

We start our analysis by defining predicates:

- On-table(x): x is placed on the table
- On(x,y): x is placed on y
- Clear(x): x does not have any object on it
- Empty-arm(x): the robotic arm is not holding any object
- Holding(x): the object x is being held by the robotic arm
- Used-cols-num(n): n block columns are being used
- Heavier(x,y): the object x weighs more than y
- Light-block(x): the weight of the object x is 1 kg

The operators as well as the preconditions, the predicates they delete and add were defined as follows:

Operator	Preconditions	Delete	Add	Description
Pick-up(x)	On-table (x) Clear (x) Empty-arm(R)	On-table (x) Empty-arm(R)	Holding(x) Used-cols-num(n=n-1)	To pick up a block "x" from the table using the right arm
Unstack (x,y)	On (x,y) Clear (x) Empty-arm(R)	On (x,y) Empty-arm(R) Clear(x)	Holding (x) Clear (y)	Picking up a block "x" that is on top of a block "y" using the right arm
Leave (x,a)	Holding (x) Used-cols-num(n<m)	Holding (x)	On-table (x) Empty-arm(a) Used-cols-nums(n=n+1)	To leave a block "x" on the table using arm "a"
Stack (x,y, a)	Holding (x) Clear (y) Heavier(y,x)	Holding (x) Clear (y)	On (x,y) Empty-arm(a) Clear(x)	To put a block "x" on top of a block "y" using "a" arm

For the left hand of the robot we have:

Operator	Preconditions	Delete	Add	Description
Unstack-Left-Hand(x,y)	On (x,y) Clear (y) Empty-arm(L) Light-block(x)	On(x,y) Empty-arm(L)	Holding(x) Clear(y)	Picking up a block "x" that is on top of a block "y" using the left arm

Pick-Up-Left-Hand(x)	Clear(x) On-table(x) Empty-arm(L) Light-block(x)	On-table(x) Empty-arm(L)	Hold(x)	To pick up a block "x" from the table using the left arm
----------------------	---	-----------------------------	---------	--

The robot has a left and right arm, where the left arm can pick up only 1kg blocks and right arm can pick up any of the blocks, this is why some operators were changed to include if it uses the left arm, because these operators will also need to have a "Light-block(x)". This precondition is only needed in the cases where the robot needs to pick up the block, for operators like Leave or Unstack it is insignificant to separate it since the arm would be already holding the block. Additionally, if we want to stack an object on top of another, the object that would be below has to be heavier than the object on top hence the Stack operator has a "Heavier(y,x)" precondition.

# Planning Algorithm

---

The program is initially loaded with information about the operators and the preconditions mentioned before, therefore it has knowledge of how the operators are loaded and how they work. On top of that, it also has a validator that checks if two or more predicates are inconsistent in a state. The rest of the information that it handles has to come from the input provided by the user, that is, the input file with: max number of columns, the blocks and their weight, the predicates of the initial state and the predicates of the goal state.

The planning algorithm consist on the following list of steps:

1. Load the input from a file introduced by the user, and map the information to system objects.
2. Validate the initial state and goal state
3. Get the list of possible operators that need to be applied to reach the goal state
4. Calculate the outcome state after applying each operator
5. Validate that the state is possible, that the state complies with the restrictions of the world and that it is not a repeated state
6. For each new state, if none of them is the initial state, repeat 3-5, but, instead of using the goal state, using the new state.
7. Once we have a state equal to the initial state, we load the output file with the information of the operators used to get there and the failed states.

In the program we created objects for the basic elements of the system: Block, State, Arm (and ArmType Enum), Predicate (and PredicateName enum), Operator (and OperatorName enum) and a InputLoader and OutputLoader to handle the reading and creation of files. Additionally, we created a Plan and Planner object, the purpose of these objects is to maintain the different Plans that are currently being tested in a way that: a Planner has a list of valid plans and failed plans, and a plan has a list of States and the operators used to get to that State.

The first step of the algorithm is executed by the InputLoader, it receives the file path of a file from user input, and (using the format established) reads the information of the system. For the second step, we validate that both the goal state and initial state are possible states, this way we make sure that we will probably be able to find a plan for this problem. The way we validate a state is as follows:

- By checking the predicates:
  - Clear(x): is invalid if On(y,x) or Holding(x) exists
  - Empty-Arm(a): is invalid if Holding(x, a) exists
  - Heavier(x,y): is invalid if On(x,y) exists
  - Holding(x,a): is invalid if Clear(x), On(x,y), On(y,x), On-table(x) or Empty-arm(a) exists
  - On(x,y): is invalid if Holding(x), Holding(y), On-table(x), Clear(y), Heavier(x,y), On(z,y) or On(y,x) exists

- On-table(x): is invalid if Holding(x,a), On(x,y) or Used-cols-num( $m > n$ ) exists
- If maximum number of columns was reached
- If it is equal to any previous state calculated

If the state doesn't comply with the validations listed before, the state is set to be invalid, if this happens with the goal or initial states, the execution stops and a warning message is shown on the console. For the fourth step, for each predicate in the current stage to be evaluated, we look for the list of operators that has that predicate in its add list, and we determine the values of the blocks that are assigned to the operator. In some cases, parameters have to be used if not all the blocks that the operator uses are known. For example, when finding the list of operators for a Clear(x) predicate, we may have Unstack(y,x), and "y" in that case is a parameter since initially we have no way of knowing what block it is. For this case, we replace the "y" with every single block so that it can be applied and the output state validated. After getting the operators, we move on to step five, where we try to apply the operator on the state. But before that, we validate that the current state has all of the predicates from the operators add list, if it does then we can apply it, if not, then the operator cannot be used for that state. To apply the operator to the current state, we first remove the predicates from the current state that matches those in the operators add list, then we add to the states predicates the preconditions of the operator, and then we clean the predicates to make sure they aren't repeated. After this, we will have a list of possible states that are the outcome of applying different operators, that list is cleaned to make sure that no operators were repeated and each of the states is validated, as it states in step 6 of the algorithm. After that, new Plan objects are created using the outcome states and operators used, these plans are added to the list of plans that the Planner object has. If in the previous step we did not find a state that is equal to the initial state, we continue calculating new operators and states for each of the plans that the Planner object has, that way, we iterate over possible plans until one of them reaches the initial state. After that, we load the output file with the information of the plan followed and listing all the failed plans that were encountered along the way.

# Implementation Design

---

The design of the implementation follows a MVC (Model, View, Controller) design pattern, where the View package has the Main class and receives the input from the user, the Controller package handles all of the business logic, and the Model package handles the basic objects and domain.

In the Model package with have the following classes:

Class name	Type	Attributes	Methods	Description
Arm	Class	- ArmType armType - int maxWeight	- getters and setters	Object to map the robots Arm
ArmType	Enum	LEFT, RIGHT, GENERIC		Object to map the type of arms. GENERIC is used for basic loading
Block	Class	- String name - int weight	-getters and setters	Object to map the blocks used
InputLoader	Static Class		-Map<String,Obje ct> loadInput(filepath )	Loads the input from the file path
Operator	Class	- OperatorName name - List<Predicate> preconditions - List<Predicate> addList - List<Predicate> delList - List<Block> paramList - ArmType arm	- getters and setters - override equals	Object to map the operators, the preconditions, addlist, dellist, blocks or arms used.
OperatorLoader	Static Class		-load(name, blockX, blockY, type) - loadOperators() - load for each operator name	Knows how to load operator given a name, blocks and an arm type.
OperatorName	Enum	STACK,UNSTACK, PICK_UP, LEAVE, UNSTACK_LEFT,		Object to map the different operators



		PICKUP_LEFT		name
OutputLoader	Static Class		- generateOutput(planner)	Generates the output file
Plan	Class	- State initialState - List<State> states - bool isValidPlan - bool isCompleted	- getters and setters - Plan makeCopy()	Handles the mapping of different plans found for the planning algorithm
Planner	Class	- List<Plan> plans - List<Plan> failedPlans - Plan completedPlan	- getters and setters	Handles all the plans that are being evaluated or have failed
Predicate	Class	- PredicateName name - List<Block> blocks - int col - ArmType arm	- getters and setters - Override equals	Object to map the predicates
PredicateName	Enum	ON, ON_TABLE, CLEAR, HOLDING, EMPTY_ARM, HEAVIER, LIGHT_BLOCK, USED_COLS_NUM		Object to map the different values of a predicates name
State	Class	- List<Predicate> predicates - Operator operatorUsedToReachState - bool isValid - String reasonForInvalid	- getters and setters - Override equals	Object that handles a state, and has information if its valid or not

All of the algorithms logic is in the BlockWorldController class, which has the following methods and attributes:

- Attributes
  - List<Block> blocks: the list of blocks in the current problem
  - Int maximumColumns: the max number of columns allowed
  - State initialState: the initial state
  - State goalState: the final state
  - List<Operator> operators: a list of all the operators available with generic params and arms
  - Planner planner: the object that holds the information of all the plans that are being processed

- List<Plan> auxPlans: an aux list for the planner.
- Methods:
  - void loadBlockWorld(String filepath): calls the input loader to load all the attributes of the class, also loads the operator list using the OperatorLoader class
  - void start(): starts the execution of the planner
  - void cyclePlanner(int count): cycles the plans in the planner until a completed plan is found
  - void calculatePlan( Plan plan): this method calculates the next possible states that can be the outcome of a plan by applying different operators.
  - List<State> removeRepeatedOperators(List<State> states): removes any repeated operators that may have been applied
  - boolean isPossibleState(State state, Plan plan, boolean onlyPred): calls all the methods that validate a state
  - boolean hasContradictingPredicates(State state): checks the state predicates and validates that there are no contradicting ones
  - boolean isEqualToAnyPreviousState(State state, Plan plan): checks if the state has already been reached before by another plan
  - List< State> calculateNewStates(State currentState): calculates all of the possible new states based on the current state
  - List<Operator> getOperatorsThatHasPredicateInAddList(Predicate predicate): gets a list of operators that has the predicate in its add list
  - List<State> stateFoundAfterOperator(State state, Operator operator, Predicate predicate): attempts to apply the operator to the state, returns the possible outcomes
  - State removeRepeatedPredicates(State state): removes all the repeated predicates of a State
  - private List<Operator> getOperatorsWithBlocks(Operator operator, Predicate predicate): loads an operator with the correct blocks
  - List<Operator> calculateValueForParam(Operator operator, int posOfParam, ArmType type): if an operator has any params, calculates the possible values for it

The class diagram was added to the zip file.

## Testing cases and results

---

For the testing cases we created basic input text files that increased in difficulty and steps, that way we could validate early on that the basics of the algorithm was working and then fix any issue that would come up. We also measured the time it took, so we could see how changes affected the algorithms execution time. All of the testing files used are contained in the zip delivered, under the resources folder.

For the first test we prepared the following text (Figure 1: testing4.txt), which would basically only have to pick up a block

```
MaxColumns=2;  
Blocks=A*;  
InitialState=ON-TABLE(A).CLEAR(A).EMPTY-ARM(L).EMPTY-ARM(R).LIGHT-BLOCK(A)  
;  
GoalState=HOLDING(A,L).EMPTY-ARM(R);
```

Figure 1: testing4.txt

This input file describes what's shown on Figure 2, where the initial state is the block A on the table, and the goal state is to pick up the block with the left arm

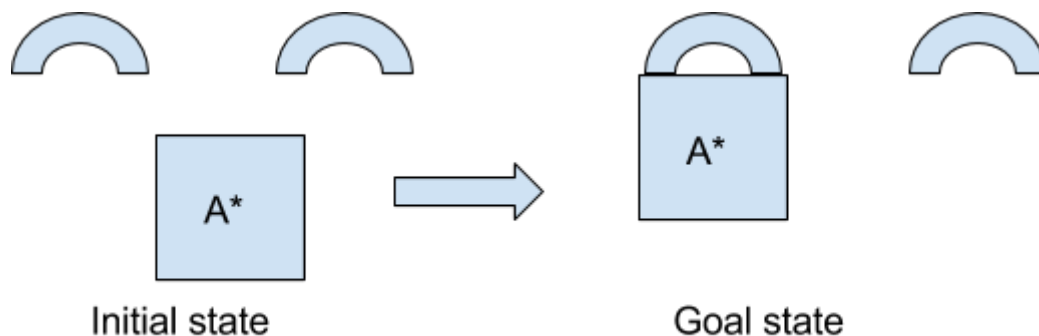


Figure 2: testing4 diagram

The output file is generated in the project folder and contains what is shown on Figure 3, and the execution output on the console showing general information and the time it took to execute it on Figure 4.

```

1
1
PICKUP_LEFT(A, Arm: LEFT)
-----

```

Figure 3: output testing4

```

<terminated> Main [Java Application] /usr/lib/jvm/java-8-oracle/bin/java (Nov 1, 2017, 1
Starting program
Please enter the file path to be read:
Filepath is: resources/testing4.txt
Map is: {GoalState=State [predicates=[Predicate [HOLDING([Block [name=A, weight=1]],
Starting planner
Goal state is valid, calculating
Planning algorithm took 78 milliseconds to find a plan
Output file is: par-activity-output1509558969723.txt

```

Figure 4: console output for testing4

The total outcome of the testing files are listed as follows with the number of steps it takes to complete each:

File	Time in ms	# steps	Input	Output	Console
testing3.txt	110	2	Figure 5	Figure 6	Figure 7
testing4.txt	78	1	Figure 3	Figure 4	Figure 5
testing5.txt	156	4	Figure 8	Figure 9	Figure 10
testing6.txt	48	2	Figure 11	Figure 12	Figure 13
testing7.txt	1290	8	Figure 14	Figure 15	Figure 16
testing2.txt	43207	12	Figure 17	Figure 18	Figure 19
testing1.txt	21494	No plan found	Figure 20	Figure 21	Figure 22

So, as we can see in the table before, the number of steps directly affects the time the planner takes to find a plan for the problem proposed. The higher the number, the more complex the problem is and the more states it has to evaluate and the more possibilities of operators that can be used. All of that causes the execution of the program to start needing more and more processing capacity to be able to handle larger problems. The algorithm will eventually need to be optimized or changed in order to execute more complex tasks, or else it will need a lot of computational power and will still take a lot of execution time to find

results, even more when you consider the fact that more operators, blocks and elements may exist.

## Instructions to execute the program

---

The program can be executed as a normal java project. Open in an IDE (IntelliJ, NetBeans, Eclipse) and run the Main.java file under the View package (src/view/Main.java).

# Appendix

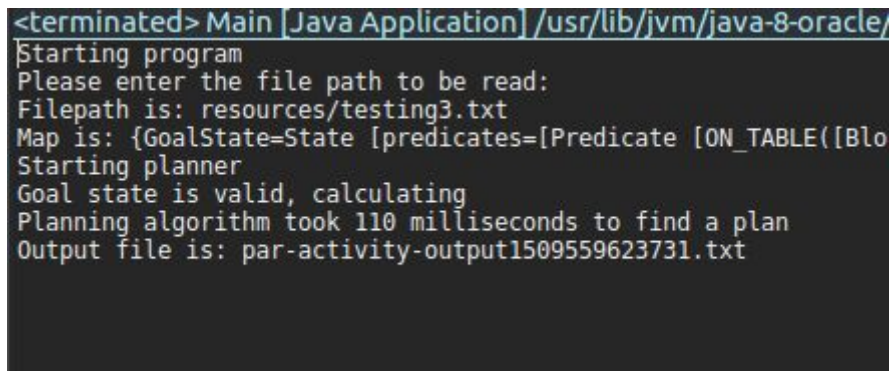
---

```
MaxColumns=2;
Blocks=A**.B*;
InitialState=ON-TABLE(A).ON(B,A).CLEAR(B).EMPTY-ARM(L).EMPTY-ARM(R);
GoalState=ON-TABLE(A).ON-TABLE(B).CLEAR(A).CLEAR(B).EMPTY-ARM(L).EMPTY-ARM(R);
```

Figure 5: testing3.txt input file

```
2
2
LEAVE(B, Arm: LEFT),UNSTACK_LEFT(B, A, Arm: LEFT)
-----
```

Figure 6: testing3.txt output file



```
<terminated> Main [Java Application] /usr/lib/jvm/java-8-oracle/
Starting program
Please enter the file path to be read:
Filepath is: resources/testing3.txt
Map is: {GoalState=State [predicates=[Predicate [ON_TABLE([Blo
Starting planner
Goal state is valid, calculating
Planning algorithm took 110 milliseconds to find a plan
Output file is: par-activity-output1509559623731.txt
```

Figure 7: testing3.txt console output

```
MaxColumns=2;
Blocks=A***.B**.C*;
InitialState=ON-TABLE(A).ON(B,A).CLEAR(B).EMPTY-ARM(L).EMPTY-ARM(R).ON-TAB
LE(C).CLEAR(C);
GoalState=ON-TABLE(A).ON-TABLE(B).CLEAR(A).CLEAR(C).ON(C,B).EMPTY-ARM(L).
EMPTY-ARM(R);
```

Figure 8: testing5.txt input file

```
4
4
STACK(C, B, Arm: LEFT),LEAVE(B, Arm: RIGHT),PICKUP_LEFT(C, Arm:
LEFT),UNSTACK(B, A, Arm: RIGHT)
-----
State:
Operators used: STACK(C, B, Arm: RIGHT),UNSTACK(C, B, Arm: RIGHT),STACK(C, B,
Arm: RIGHT)
Predicate [ON_TABLE([Block [name=A, weight=3]])],Predicate [ON_TABLE([Block
[name=B, weight=2]])],Predicate [CLEAR([Block [name=A, weight=3]])],Predicate
[EMPTY_ARM(LEFT)],Predicate [HEAVIER([Block [name=B, weight=2], Block [name=C,
weight=1]])],Predicate [CLEAR([Block [name=B, weight=2]])],Predicate [HOLDING([Block
[name=C, weight=1], RIGHT)],Predicate [HEAVIER([Block [name=B, weight=2], Block
[name=C, weight=1]])]
State is equal to another previous state already found
-----
State:
Operators used: STACK(C, B, Arm: RIGHT),UNSTACK(C, A, Arm: RIGHT),STACK(C, A,
Arm: RIGHT)
Predicate [ON_TABLE([Block [name=A, weight=3]])],Predicate [ON_TABLE([Block
[name=B, weight=2]])],Predicate [EMPTY_ARM(LEFT)],Predicate [CLEAR([Block
[name=B, weight=2]])],Predicate [HEAVIER([Block [name=B, weight=2], Block [name=C,
weight=1]])],Predicate [CLEAR([Block [name=A, weight=3]])],Predicate [HOLDING([Block
[name=C, weight=1], RIGHT)],Predicate [HEAVIER([Block [name=A, weight=3], Block
[name=C, weight=1]])]
State is equal to another previous state already found
-----
State:
Operators used: STACK(C, B, Arm: LEFT),UNSTACK_LEFT(C, A, Arm: LEFT),STACK(C,
A, Arm: LEFT)
Predicate [ON_TABLE([Block [name=A, weight=3]])],Predicate [ON_TABLE([Block
[name=B, weight=2]])],Predicate [EMPTY_ARM(RIGHT)],Predicate [CLEAR([Block
[name=B, weight=2]])],Predicate [HEAVIER([Block [name=B, weight=2], Block [name=C,
weight=1]])],Predicate [LIGHT_BLOCK([Block [name=C, weight=1]])],Predicate
[CLEAR([Block [name=A, weight=3]])],Predicate [HOLDING([Block [name=C, weight=1],
LEFT)],Predicate [HEAVIER([Block [name=A, weight=3], Block [name=C, weight=1]])]
State is equal to another previous state already found
-----
State:
Operators used: STACK(C, B, Arm: LEFT),LEAVE(A, Arm: RIGHT),PICK_UP(A, Arm:
RIGHT)
Predicate [ON_TABLE([Block [name=B, weight=2]])],Predicate [CLEAR([Block [name=B,
weight=2]])],Predicate [HOLDING([Block [name=C, weight=1], LEFT)],Predicate
[HEAVIER([Block [name=B, weight=2], Block [name=C, weight=1]])],Predicate
[EMPTY_ARM(RIGHT)],Predicate [CLEAR([Block [name=A, weight=3]])],Predicate
[ON_TABLE([Block [name=A, weight=3]])]
State is equal to another previous state already found
```



```

-----
State:
Operators used: STACK(C, B, Arm: LEFT),UNSTACK_LEFT(C, B, Arm: LEFT),STACK(C,
B, Arm: LEFT)
Predicate [ON_TABLE([Block [name=A, weight=3]])],Predicate [ON_TABLE([Block
[name=B, weight=2]])],Predicate [CLEAR([Block [name=A, weight=3]])],Predicate
[EMPTY_ARM(RIGHT)],Predicate [HEAVIER([Block [name=B, weight=2], Block [name=C,
weight=1]])],Predicate [LIGHT_BLOCK([Block [name=C, weight=1]])],Predicate
[CLEAR([Block [name=B, weight=2]])],Predicate [HOLDING([Block [name=C, weight=1],
LEFT]),Predicate [HEAVIER([Block [name=B, weight=2], Block [name=C, weight=1]])]
State is equal to another previous state already found
-----
State:
Operators used: STACK(C, B, Arm: LEFT),LEAVE(B, Arm: RIGHT),PICK_UP(B, Arm:
RIGHT)
Predicate [ON_TABLE([Block [name=A, weight=3]])],Predicate [CLEAR([Block [name=A,
weight=3]])],Predicate [HOLDING([Block [name=C, weight=1], LEFT]),Predicate
[HEAVIER([Block [name=B, weight=2], Block [name=C, weight=1]])],Predicate
[EMPTY_ARM(RIGHT)],Predicate [CLEAR([Block [name=B, weight=2]])],Predicate
[ON_TABLE([Block [name=B, weight=2]])]
State is equal to another previous state already found
-----
State:
Operators used: STACK(C, B, Arm: RIGHT),UNSTACK(C, B, Arm: RIGHT),STACK(C, B,
Arm: RIGHT)
Predicate [ON_TABLE([Block [name=A, weight=3]])],Predicate [ON_TABLE([Block
[name=B, weight=2]])],Predicate [CLEAR([Block [name=A, weight=3]])],Predicate
[EMPTY_ARM(LEFT)],Predicate [HEAVIER([Block [name=B, weight=2], Block [name=C,
weight=1]])],Predicate [CLEAR([Block [name=B, weight=2]])],Predicate [HOLDING([Block
[name=C, weight=1], RIGHT]),Predicate [HEAVIER([Block [name=B, weight=2], Block
[name=C, weight=1]])]
State is equal to another previous state already found
** FILE SHORTENED **

```

Figure 9: testing5.txt output file

```

<terminated> Main [Java Application] /usr/lib/jvm/java-8-oracle/bin/java (Nov 1, 2017,
Starting program
Please enter the file path to be read:
Filepath is: resources/testing5.txt
Map is: {GoalState=State [predicates=[Predicate [ON_TABLE([Block [name=A, weight=3]]
Starting planner
Goal state is valid, calculating
Planning algorithm took 156 milliseconds to find a plan
Output file is: par-activity-output1509559685864.txt

```

Figure 10: testing5.txt console output

```

MaxColumns=2;
Blocks=A**.B*.C**;
InitialState=ON-TABLE(A).ON(B,A).CLEAR(B).EMPTY-ARM(L).EMPTY-ARM(R).ON-TAB
LE(C).CLEAR(C);
GoalState=ON-TABLE(A).ON-TABLE(C).CLEAR(A).ON(B,C).EMPTY-ARM(L).EMPTY-AR
M(R).CLEAR(B);

```

Figure 11: testing6.txt input file

```

2
2
STACK(B, C, Arm: RIGHT),UNSTACK(B, A, Arm: RIGHT)
-----

```

Figure 12: testing6.txt output file

```

<terminated> Main [Java Application] /usr/lib/jvm/java-8-oracle/bin/java (Nov 1
Starting program
Please enter the file path to be read:
Filepath is: resources/testing6.txt
Map is: {GoalState=State [predicates=[Predicate [ON_TABLE([Block [name=A, wei
Starting planner
Goal state is valid, calculating
Planning algorithm took 48 milliseconds to find a plan
Output file is: par-activity-output1509559780037.txt

```

Figure 13: testing6.txt console output

```

MaxColumns=2;
Blocks=A*****.B**.C*.D*;
InitialState=ON-TABLE(A).ON(B,A).ON(C,B).CLEAR(C).EMPTY-ARM(L).EMPTY-ARM(R)
.ON-TABLE(D).CLEAR(D);
GoalState=ON-TABLE(A).ON-TABLE(B).CLEAR(B).ON(D,A).ON(C,D).EMPTY-ARM(L).E
MPTY-ARM(R).CLEAR(C);

```

Figure 14: testing7.txt input file

```

8
8

```

STACK(C, D, Arm: LEFT),LEAVE(B, Arm: RIGHT),PICKUP\_LEFT(C, Arm: LEFT),STACK(D, A, Arm: LEFT),UNSTACK(B, A, Arm: RIGHT),LEAVE(C, Arm: RIGHT),PICKUP\_LEFT(D, Arm: LEFT),UNSTACK(C, B, Arm: RIGHT)

---

State:

Operators used: STACK(B, A, Arm: RIGHT)

Predicate [ON\_TABLE([Block [name=A, weight=5]])],Predicate [ON\_TABLE([Block [name=B, weight=2]])],Predicate [ON([Block [name=D, weight=1], Block [name=A, weight=5]])],Predicate [ON([Block [name=C, weight=1], Block [name=D, weight=1]])],Predicate [EMPTY\_ARM(LEFT)],Predicate [CLEAR([Block [name=C, weight=1]])],Predicate [CLEAR([Block [name=A, weight=5]])],Predicate [HOLDING([Block [name=B, weight=2], RIGHT]),Predicate [HEAVIER([Block [name=A, weight=5], Block [name=B, weight=2]])]

For operator: [STACK(B, A, Arm: RIGHT)] to be used State needed to have: Predicate [ON([Block [name=B, weight=2], Block [name=A, weight=5]])]

---

State:

Operators used: STACK(B, C, Arm: RIGHT)

Predicate [ON\_TABLE([Block [name=A, weight=5]])],Predicate [ON\_TABLE([Block [name=B, weight=2]])],Predicate [ON([Block [name=D, weight=1], Block [name=A, weight=5]])],Predicate [ON([Block [name=C, weight=1], Block [name=D, weight=1]])],Predicate [EMPTY\_ARM(LEFT)],Predicate [CLEAR([Block [name=C, weight=1]])],Predicate [CLEAR([Block [name=C, weight=1]])],Predicate [HOLDING([Block [name=B, weight=2], RIGHT]),Predicate [HEAVIER([Block [name=C, weight=1], Block [name=B, weight=2]])]

For operator: [STACK(B, C, Arm: RIGHT)] to be used State needed to have: Predicate [ON([Block [name=B, weight=2], Block [name=C, weight=1]])]

---

State:

Operators used: STACK(B, D, Arm: RIGHT)

Predicate [ON\_TABLE([Block [name=A, weight=5]])],Predicate [ON\_TABLE([Block [name=B, weight=2]])],Predicate [ON([Block [name=D, weight=1], Block [name=A, weight=5]])],Predicate [ON([Block [name=C, weight=1], Block [name=D, weight=1]])],Predicate [EMPTY\_ARM(LEFT)],Predicate [CLEAR([Block [name=C, weight=1]])],Predicate [CLEAR([Block [name=D, weight=1]])],Predicate [HOLDING([Block [name=B, weight=2], RIGHT]),Predicate [HEAVIER([Block [name=D, weight=1], Block [name=B, weight=2]])]

For operator: [STACK(B, D, Arm: RIGHT)] to be used State needed to have: Predicate [ON([Block [name=B, weight=2], Block [name=D, weight=1]])]

---

State:

Operators used: UNSTACK(A, B, Arm: RIGHT)

Predicate [ON\_TABLE([Block [name=A, weight=5]])],Predicate [ON\_TABLE([Block

[name=B, weight=2]]],Predicate [ON([Block [name=D, weight=1], Block [name=A, weight=5]]],Predicate [ON([Block [name=C, weight=1], Block [name=D, weight=1]]],Predicate [EMPTY\_ARM(LEFT)],Predicate [CLEAR([Block [name=C, weight=1]]],Predicate [EMPTY\_ARM(RIGHT)],Predicate [CLEAR([Block [name=A, weight=5]]],Predicate [ON([Block [name=A, weight=5], Block [name=B, weight=2]])]  
Predicate: [Predicate [ON([Block [name=A, weight=5], Block [name=B, weight=2]])]]  
impossible because of the weight of the blocks

-----

State:

Operators used: UNSTACK(D, B, Arm: RIGHT)

Predicate [ON\_TABLE([Block [name=A, weight=5]]],Predicate [ON\_TABLE([Block [name=B, weight=2]]],Predicate [ON([Block [name=D, weight=1], Block [name=A, weight=5]]],Predicate [ON([Block [name=C, weight=1], Block [name=D, weight=1]]],Predicate [EMPTY\_ARM(LEFT)],Predicate [CLEAR([Block [name=C, weight=1]]],Predicate [EMPTY\_ARM(RIGHT)],Predicate [CLEAR([Block [name=D, weight=1]]],Predicate [ON([Block [name=D, weight=1], Block [name=B, weight=2]])]  
Found contradicting predicates: [Predicate [CLEAR([Block [name=D, weight=1]])]] with:  
[Predicate [ON([Block [name=C, weight=1], Block [name=D, weight=1]])]]

-----

State:

Operators used: UNSTACK\_LEFT(A, B, Arm: LEFT)

Predicate [ON\_TABLE([Block [name=A, weight=5]]],Predicate [ON\_TABLE([Block [name=B, weight=2]]],Predicate [ON([Block [name=D, weight=1], Block [name=A, weight=5]]],Predicate [ON([Block [name=C, weight=1], Block [name=D, weight=1]]],Predicate [EMPTY\_ARM(RIGHT)],Predicate [CLEAR([Block [name=C, weight=1]]],Predicate [EMPTY\_ARM(LEFT)],Predicate [CLEAR([Block [name=A, weight=5]]],Predicate [ON([Block [name=A, weight=5], Block [name=B, weight=2]])],Predicate [LIGHT\_BLOCK([Block [name=A, weight=5]])]  
Predicate: [Predicate [ON([Block [name=A, weight=5], Block [name=B, weight=2]])]]  
impossible because of the weight of the blocks

-----

State:

Operators used: UNSTACK\_LEFT(D, B, Arm: LEFT)

Predicate [ON\_TABLE([Block [name=A, weight=5]]],Predicate [ON\_TABLE([Block [name=B, weight=2]]],Predicate [ON([Block [name=D, weight=1], Block [name=A, weight=5]]],Predicate [ON([Block [name=C, weight=1], Block [name=D, weight=1]]],Predicate [EMPTY\_ARM(RIGHT)],Predicate [CLEAR([Block [name=C, weight=1]]],Predicate [EMPTY\_ARM(LEFT)],Predicate [CLEAR([Block [name=D, weight=1]]],Predicate [ON([Block [name=D, weight=1], Block [name=B, weight=2]])],Predicate [LIGHT\_BLOCK([Block [name=D, weight=1]])]  
Found contradicting predicates: [Predicate [CLEAR([Block [name=D, weight=1]])]] with:  
[Predicate [ON([Block [name=C, weight=1], Block [name=D, weight=1]])]]

-----

\*\*FILE SHORTENED\*\*

Figure 15: testing7.txt output file

```
<terminated>> Main [Java Application] /usr/lib/jvm/java-8-oracle/bin/java (Nov 1, 2017)
Starting program
Please enter the file path to be read:
Filepath is: resources/testing7.txt
Map is: {GoalState=State [predicates=[Predicate [ON_TABLE([Block [name=A, weight=9], name=B, weight=6]), Block [name=C, weight=1]], GoalTestFunction=GoalTestFunction]]}
Starting planner
Goal state is valid, calculating
Planning algorithm took 1290 milliseconds to find a plan
Output file is: par-activity-output1509559824694.txt
```

Figure 16: testing7.txt console output

```
MaxColumns=3;
Blocks=A*.B**.C**.E****.F*;
InitialState=ON-TABLE(E).ON(C,E).ON(B,C).CLEAR(B).ON-TABLE(F).ON(A,F).CLEAR(A)
).EMPTY-ARM(L).EMPTY-ARM(R);
GoalState=ON-TABLE(E).ON(B,E).ON(A,B).CLEAR(A).ON-TABLE(C).ON(F,C).CLEAR(F)
).EMPTY-ARM(L).EMPTY-ARM(R);
```

Figure 17: testing2.txt input file

```

12
12
STACK(A, B, Arm: LEFT),STACK(B, E, Arm: RIGHT),UNSTACK_LEFT(A, E, Arm:
LEFT),PICK_UP(B, Arm: RIGHT),STACK(F, C, Arm: LEFT),LEAVE(C, Arm:
RIGHT),PICKUP_LEFT(F, Arm: LEFT),STACK(A, E, Arm: LEFT),UNSTACK(C, E, Arm:
RIGHT),LEAVE(B, Arm: RIGHT),UNSTACK_LEFT(A, F, Arm: LEFT),UNSTACK(B, C,
Arm: RIGHT)
-----
State:
Operators used: LEAVE(C, Arm: RIGHT)
Predicate [ON_TABLE([Block [name=E, weight=4]])],Predicate [ON([Block [name=B,
weight=2], Block [name=E, weight=4]])],Predicate [ON([Block [name=A, weight=1], Block
[name=B, weight=2]])],Predicate [CLEAR([Block [name=A, weight=1]])],Predicate
[ON([Block [name=F, weight=1], Block [name=C, weight=2]])],Predicate [CLEAR([Block

```

[name=F, weight=1]]],Predicate [EMPTY\_ARM(LEFT)],Predicate [HOLDING([Block [name=C, weight=2]], RIGHT)]

For operator: [LEAVE(C, Arm: RIGHT)] to be used State needed to have: Predicate [CLEAR([Block [name=C, weight=2]])]

-----

State:

Operators used: LEAVE(E, Arm: RIGHT)

Predicate [ON([Block [name=B, weight=2], Block [name=E, weight=4]])],Predicate [ON([Block [name=A, weight=1], Block [name=B, weight=2]])],Predicate [CLEAR([Block [name=A, weight=1]])],Predicate [ON\_TABLE([Block [name=C, weight=2]])],Predicate [ON([Block [name=F, weight=1], Block [name=C, weight=2]])],Predicate [CLEAR([Block [name=F, weight=1]])],Predicate [EMPTY\_ARM(LEFT)],Predicate [HOLDING([Block [name=E, weight=4]], RIGHT)]

For operator: [LEAVE(E, Arm: RIGHT)] to be used State needed to have: Predicate [CLEAR([Block [name=E, weight=4]])]

-----

State:

Operators used: STACK(A, C, Arm: RIGHT)

Predicate [ON\_TABLE([Block [name=E, weight=4]])],Predicate [ON([Block [name=B, weight=2], Block [name=E, weight=4]])],Predicate [ON([Block [name=A, weight=1], Block [name=B, weight=2]])],Predicate [ON\_TABLE([Block [name=C, weight=2]])],Predicate [ON([Block [name=F, weight=1], Block [name=C, weight=2]])],Predicate [CLEAR([Block [name=F, weight=1]])],Predicate [EMPTY\_ARM(LEFT)],Predicate [CLEAR([Block [name=C, weight=2]])],Predicate [HOLDING([Block [name=A, weight=1]], RIGHT)],Predicate [HEAVIER([Block [name=C, weight=2], Block [name=A, weight=1]])]

For operator: [STACK(A, C, Arm: RIGHT)] to be used State needed to have: Predicate [ON([Block [name=A, weight=1], Block [name=C, weight=2]])]

-----

State:

Operators used: STACK(A, E, Arm: RIGHT)

Predicate [ON\_TABLE([Block [name=E, weight=4]])],Predicate [ON([Block [name=B, weight=2], Block [name=E, weight=4]])],Predicate [ON([Block [name=A, weight=1], Block [name=B, weight=2]])],Predicate [ON\_TABLE([Block [name=C, weight=2]])],Predicate [ON([Block [name=F, weight=1], Block [name=C, weight=2]])],Predicate [CLEAR([Block [name=F, weight=1]])],Predicate [EMPTY\_ARM(LEFT)],Predicate [CLEAR([Block [name=E, weight=4]])],Predicate [HOLDING([Block [name=A, weight=1]], RIGHT)],Predicate [HEAVIER([Block [name=E, weight=4], Block [name=A, weight=1]])]

For operator: [STACK(A, E, Arm: RIGHT)] to be used State needed to have: Predicate [ON([Block [name=A, weight=1], Block [name=E, weight=4]])]

-----

State:

Operators used: STACK(A, F, Arm: RIGHT)

Predicate [ON\_TABLE([Block [name=E, weight=4]])],Predicate [ON([Block [name=B,

weight=2], Block [name=E, weight=4]]], Predicate [ON([Block [name=A, weight=1], Block [name=B, weight=2]])], Predicate [ON\_TABLE([Block [name=C, weight=2]])], Predicate [ON([Block [name=F, weight=1], Block [name=C, weight=2]])], Predicate [CLEAR([Block [name=F, weight=1]])], Predicate [EMPTY\_ARM(LEFT)], Predicate [CLEAR([Block [name=F, weight=1]])], Predicate [HOLDING([Block [name=A, weight=1], RIGHT]), Predicate [HEAVIER([Block [name=F, weight=1], Block [name=A, weight=1]])]

For operator: [STACK(A, F, Arm: RIGHT)] to be used State needed to have: Predicate [ON([Block [name=A, weight=1], Block [name=F, weight=1]])]

-----

State:

Operators used: STACK(B, A, Arm: RIGHT)

Predicate [ON\_TABLE([Block [name=E, weight=4]])], Predicate [ON([Block [name=B, weight=2], Block [name=E, weight=4]])], Predicate [ON([Block [name=A, weight=1], Block [name=B, weight=2]])], Predicate [CLEAR([Block [name=A, weight=1]])], Predicate [ON\_TABLE([Block [name=C, weight=2]])], Predicate [ON([Block [name=F, weight=1], Block [name=C, weight=2]])], Predicate [CLEAR([Block [name=F, weight=1]])], Predicate [EMPTY\_ARM(LEFT)], Predicate [CLEAR([Block [name=A, weight=1]])], Predicate [HOLDING([Block [name=B, weight=2], RIGHT]), Predicate [HEAVIER([Block [name=A, weight=1], Block [name=B, weight=2]])]

For operator: [STACK(B, A, Arm: RIGHT)] to be used State needed to have: Predicate [ON([Block [name=B, weight=2], Block [name=A, weight=1]])]

-----

State:

Operators used: STACK(B, C, Arm: RIGHT)

Predicate [ON\_TABLE([Block [name=E, weight=4]])], Predicate [ON([Block [name=B, weight=2], Block [name=E, weight=4]])], Predicate [ON([Block [name=A, weight=1], Block [name=B, weight=2]])], Predicate [CLEAR([Block [name=A, weight=1]])], Predicate [ON\_TABLE([Block [name=C, weight=2]])], Predicate [ON([Block [name=F, weight=1], Block [name=C, weight=2]])], Predicate [CLEAR([Block [name=F, weight=1]])], Predicate [EMPTY\_ARM(LEFT)], Predicate [CLEAR([Block [name=C, weight=2]])], Predicate [HOLDING([Block [name=B, weight=2], RIGHT]), Predicate [HEAVIER([Block [name=C, weight=2], Block [name=B, weight=2]])]

For operator: [STACK(B, C, Arm: RIGHT)] to be used State needed to have: Predicate [ON([Block [name=B, weight=2], Block [name=C, weight=2]])]

-----

State:

Operators used: STACK(B, E, Arm: RIGHT)

Predicate [ON\_TABLE([Block [name=E, weight=4]])], Predicate [ON([Block [name=A, weight=1], Block [name=B, weight=2]])], Predicate [CLEAR([Block [name=A, weight=1]])], Predicate [ON\_TABLE([Block [name=C, weight=2]])], Predicate [ON([Block [name=F, weight=1], Block [name=C, weight=2]])], Predicate [CLEAR([Block [name=F, weight=1]])], Predicate [EMPTY\_ARM(LEFT)], Predicate [CLEAR([Block [name=E, weight=4]])], Predicate [HOLDING([Block [name=B, weight=2], RIGHT]), Predicate

[HEAVIER([Block [name=E, weight=4], Block [name=B, weight=2]])]

For operator: [STACK(B, E, Arm: RIGHT)] to be used State needed to have: Predicate  
[CLEAR([Block [name=B, weight=2]])]

-----

State:

Operators used: STACK(B, F, Arm: RIGHT)

Predicate [ON\_TABLE([Block [name=E, weight=4]])], Predicate [ON([Block [name=B, weight=2], Block [name=E, weight=4]])], Predicate [ON([Block [name=A, weight=1], Block [name=B, weight=2]])], Predicate [CLEAR([Block [name=A, weight=1]])], Predicate [ON\_TABLE([Block [name=C, weight=2]])], Predicate [ON([Block [name=F, weight=1], Block [name=C, weight=2]])], Predicate [CLEAR([Block [name=F, weight=1]])], Predicate [EMPTY\_ARM(LEFT)], Predicate [CLEAR([Block [name=F, weight=1]])], Predicate [HOLDING([Block [name=B, weight=2], RIGHT]), Predicate [HEAVIER([Block [name=F, weight=1], Block [name=B, weight=2]])]

For operator: [STACK(B, F, Arm: RIGHT)] to be used State needed to have: Predicate  
[ON([Block [name=B, weight=2], Block [name=F, weight=1]])]

-----

State:

Operators used: STACK(C, A, Arm: RIGHT)

Predicate [ON\_TABLE([Block [name=E, weight=4]])], Predicate [ON([Block [name=B, weight=2], Block [name=E, weight=4]])], Predicate [ON([Block [name=A, weight=1], Block [name=B, weight=2]])], Predicate [CLEAR([Block [name=A, weight=1]])], Predicate [ON\_TABLE([Block [name=C, weight=2]])], Predicate [ON([Block [name=F, weight=1], Block [name=C, weight=2]])], Predicate [CLEAR([Block [name=F, weight=1]])], Predicate [EMPTY\_ARM(LEFT)], Predicate [CLEAR([Block [name=A, weight=1]])], Predicate [HOLDING([Block [name=C, weight=2], RIGHT]), Predicate [HEAVIER([Block [name=A, weight=1], Block [name=C, weight=2]])]

For operator: [STACK(C, A, Arm: RIGHT)] to be used State needed to have: Predicate  
[ON([Block [name=C, weight=2], Block [name=A, weight=1]])]

-----

State:

Operators used: STACK(C, B, Arm: RIGHT)

Predicate [ON\_TABLE([Block [name=E, weight=4]])], Predicate [ON([Block [name=B, weight=2], Block [name=E, weight=4]])], Predicate [ON([Block [name=A, weight=1], Block [name=B, weight=2]])], Predicate [CLEAR([Block [name=A, weight=1]])], Predicate [ON\_TABLE([Block [name=C, weight=2]])], Predicate [ON([Block [name=F, weight=1], Block [name=C, weight=2]])], Predicate [CLEAR([Block [name=F, weight=1]])], Predicate [EMPTY\_ARM(LEFT)], Predicate [CLEAR([Block [name=B, weight=2]])], Predicate [HOLDING([Block [name=C, weight=2], RIGHT]), Predicate [HEAVIER([Block [name=B, weight=2], Block [name=C, weight=2]])]

For operator: [STACK(C, B, Arm: RIGHT)] to be used State needed to have: Predicate  
[ON([Block [name=C, weight=2], Block [name=B, weight=2]])]

-----



\*\*FILE SHORTENED\*\*

Figure 18: testing2.txt output file

```
Starting program  
Please enter the file path to be read:  
Filepath is: resources/testing2.txt  
Map is: {GoalState=State [predicates=[Predicate [ON_TABLE([Block [name=E, weight=10], Block [name=F, weight=10]]], Block [name=G, weight=10]]]  
Starting planner  
Goal state is valid, calculating  
Planning algorithm took 43207 milliseconds to find a plan  
Output file is: par-activity-output1509568871515.txt
```

Figure 19: testing2.txt console output

```
MaxColumns=3;  
Blocks=A*.B**.C**.D***.F*;  
InitialState=ON-TABLE(C).ON(B,C).ON(A,B).CLEAR(A).ON-TABLE(D).ON(F,D).CLEAR(F  
)EMPTY-ARM(L).EMPTY-ARM(R);  
GoalState=ON-TABLE(B).ON(C,B).CLEAR(C).ON-TABLE(D).ON(A,D).ON(F,A).CLEAR(F  
)EMPTY-ARM(L).EMPTY-ARM(R);
```

Figure 20: testing1.txt input file

No plan found

---

State:

Operators used: STACK(C, B, Arm: RIGHT), PICK\_UP(C, Arm: RIGHT), LEAVE(B, Arm: RIGHT), STACK(F, A, Arm: LEFT), UNSTACK\_LEFT(F, C, Arm: LEFT), STACK(A, D, Arm: LEFT), PICKUP\_LEFT(A, Arm: LEFT), STACK(F, C, Arm: LEFT), UNSTACK(B, D, Arm: RIGHT), LEAVE(C, Arm: RIGHT), UNSTACK(C, B, Arm: RIGHT), LEAVE(A, Arm: RIGHT), UNSTACK\_LEFT(F, C, Arm: LEFT), STACK(F, C, Arm: LEFT)

Predicate [ON\_TABLE([Block [name=D, weight=3]])], Predicate [HEAVIER([Block [name=B, weight=2], Block [name=C, weight=2]])], Predicate [HEAVIER([Block [name=A, weight=1], Block [name=F, weight=1]])], Predicate [LIGHT\_BLOCK([Block [name=F, weight=1]])], Predicate [HEAVIER([Block [name=D, weight=3], Block [name=A, weight=1]])], Predicate [LIGHT\_BLOCK([Block [name=A, weight=1]])], Predicate [HEAVIER([Block [name=C, weight=2], Block [name=F, weight=1]])], Predicate [ON([Block

No plan found

---

State:

Operators used: STACK(C, B, Arm: RIGHT), PICK\_UP(C, Arm: RIGHT), LEAVE(B, Arm: RIGHT), STACK(F, A, Arm: LEFT), UNSTACK\_LEFT(F, C, Arm: LEFT), STACK(A, D, Arm: LEFT), PICKUP\_LEFT(A, Arm: LEFT), STACK(F, C, Arm: LEFT), UNSTACK(B, D, Arm: RIGHT), LEAVE(C, Arm: RIGHT), UNSTACK(C, B, Arm: RIGHT), LEAVE(A, Arm: RIGHT), UNSTACK\_LEFT(F, C, Arm: LEFT), STACK(F, C, Arm: LEFT)

Predicate [ON\_TABLE([Block [name=D, weight=3]])], Predicate [HEAVIER([Block [name=B, weight=2], Block [name=C, weight=2]])], Predicate [HEAVIER([Block [name=A, weight=1], Block [name=F, weight=1]])], Predicate [LIGHT\_BLOCK([Block [name=F, weight=1]])], Predicate [HEAVIER([Block [name=D, weight=3], Block [name=A, weight=1]])], Predicate [LIGHT\_BLOCK([Block [name=A, weight=1]])], Predicate [HEAVIER([Block [name=C, weight=2], Block [name=F, weight=1]])], Predicate [ON([Block

No plan found

---

State:

Operators used: STACK(C, B, Arm: RIGHT), PICK\_UP(C, Arm: RIGHT), LEAVE(B, Arm: RIGHT), STACK(F, A, Arm: LEFT), UNSTACK\_LEFT(F, C, Arm: LEFT), STACK(A, D, Arm: LEFT), PICKUP\_LEFT(A, Arm: LEFT), STACK(F, C, Arm: LEFT), UNSTACK(B, D, Arm: RIGHT), LEAVE(C, Arm: RIGHT), UNSTACK(C, B, Arm: RIGHT), LEAVE(A, Arm: RIGHT), UNSTACK\_LEFT(F, C, Arm: LEFT), STACK(F, C, Arm: LEFT)

Predicate [ON\_TABLE([Block [name=D, weight=3]])], Predicate [HEAVIER([Block [name=B, weight=2], Block [name=C, weight=2]])], Predicate [HEAVIER([Block [name=A, weight=1], Block [name=F, weight=1]])], Predicate [LIGHT\_BLOCK([Block [name=F, weight=1]])], Predicate [HEAVIER([Block [name=D, weight=3], Block [name=A, weight=1]])], Predicate [LIGHT\_BLOCK([Block [name=A, weight=1]])], Predicate [HEAVIER([Block [name=C, weight=2], Block [name=F, weight=1]])], Predicate [ON([Block

No plan found

---

State:

Operators used: STACK(C, B, Arm: RIGHT), PICK\_UP(C, Arm: RIGHT), LEAVE(B, Arm: RIGHT), STACK(F, A, Arm: LEFT), UNSTACK\_LEFT(F, C, Arm: LEFT), STACK(A, D, Arm: LEFT), PICKUP\_LEFT(A, Arm: LEFT), STACK(F, C, Arm: LEFT), UNSTACK(B, D, Arm: RIGHT), LEAVE(C, Arm: RIGHT), UNSTACK(C, B, Arm: RIGHT), LEAVE(A, Arm: RIGHT), UNSTACK\_LEFT(F, C, Arm: LEFT), STACK(F, C, Arm: LEFT)

Predicate [ON\_TABLE([Block [name=D, weight=3]])], Predicate [HEAVIER([Block [name=B, weight=2], Block [name=C, weight=2]])], Predicate [HEAVIER([Block [name=A, weight=1], Block [name=F, weight=1]])], Predicate [LIGHT\_BLOCK([Block [name=F, weight=1]])], Predicate [HEAVIER([Block [name=D, weight=3], Block [name=A, weight=1]])], Predicate [LIGHT\_BLOCK([Block [name=A, weight=1]])], Predicate [HEAVIER([Block [name=C, weight=2], Block [name=F, weight=1]])], Predicate [ON([Block

[name=B, weight=2], Block [name=D, weight=3]]), Predicate [ON([Block [name=C, weight=2], Block [name=B, weight=2]])], Predicate [HOLDING([Block [name=A, weight=1], RIGHT]), Predicate [LIGHT\_BLOCK([Block [name=F, weight=1]])], Predicate [CLEAR([Block [name=C, weight=2]])], Predicate [HOLDING([Block [name=F, weight=1], LEFT]), Predicate [HEAVIER([Block [name=C, weight=2], Block [name=F, weight=1]])]  
State is equal to another previous state already found

-----  
State:

Operators used: STACK(A, D, Arm: RIGHT)

Predicate [ON\_TABLE([Block [name=B, weight=2]])], Predicate [ON([Block [name=C, weight=2], Block [name=B, weight=2]])], Predicate [CLEAR([Block [name=C, weight=2]])], Predicate [ON\_TABLE([Block [name=D, weight=3]])], Predicate [ON([Block [name=F, weight=1], Block [name=A, weight=1]])], Predicate [CLEAR([Block [name=F, weight=1]])], Predicate [EMPTY\_ARM(LEFT)], Predicate [CLEAR([Block [name=D, weight=3]])], Predicate [HOLDING([Block [name=A, weight=1], RIGHT]), Predicate [HEAVIER([Block [name=D, weight=3], Block [name=A, weight=1]])]

For operator: [STACK(A, D, Arm: RIGHT)] to be used State needed to have: Predicate [CLEAR([Block [name=A, weight=1]])]

-----  
State:

Operators used: STACK(A, D, Arm: LEFT)

Predicate [ON\_TABLE([Block [name=B, weight=2]])], Predicate [ON([Block [name=C, weight=2], Block [name=B, weight=2]])], Predicate [CLEAR([Block [name=C, weight=2]])], Predicate [ON\_TABLE([Block [name=D, weight=3]])], Predicate [ON([Block [name=F, weight=1], Block [name=A, weight=1]])], Predicate [CLEAR([Block [name=F, weight=1]])], Predicate [EMPTY\_ARM(RIGHT)], Predicate [CLEAR([Block [name=D, weight=3]])], Predicate [HOLDING([Block [name=A, weight=1], LEFT]), Predicate [HEAVIER([Block [name=D, weight=3], Block [name=A, weight=1]])]

For operator: [STACK(A, D, Arm: LEFT)] to be used State needed to have: Predicate [CLEAR([Block [name=A, weight=1]])]

-----  
State:

Operators used: STACK(C, A, Arm: RIGHT)

Predicate [ON\_TABLE([Block [name=B, weight=2]])], Predicate [ON([Block [name=C, weight=2], Block [name=B, weight=2]])], Predicate [ON\_TABLE([Block [name=D, weight=3]])], Predicate [ON([Block [name=A, weight=1], Block [name=D, weight=3]])], Predicate [ON([Block [name=F, weight=1], Block [name=A, weight=1]])], Predicate [CLEAR([Block [name=F, weight=1]])], Predicate [EMPTY\_ARM(LEFT)], Predicate [CLEAR([Block [name=A, weight=1]])], Predicate [HOLDING([Block [name=C, weight=2], RIGHT]), Predicate [HEAVIER([Block [name=A, weight=1], Block [name=C, weight=2]])]

For operator: [STACK(C, A, Arm: RIGHT)] to be used State needed to have: Predicate [ON([Block [name=C, weight=2], Block [name=A, weight=1]])]

-----  
State:

Operators used: STACK(C, D, Arm: RIGHT)

Predicate [ON\_TABLE([Block [name=B, weight=2]])], Predicate [ON([Block [name=C, weight=2], Block [name=B, weight=2]])], Predicate [ON\_TABLE([Block [name=D, weight=3]])], Predicate [ON([Block [name=A, weight=1], Block [name=D, weight=3]])], Predicate [ON([Block [name=F, weight=1], Block [name=A, weight=1]])], Predicate [CLEAR([Block [name=F, weight=1]])], Predicate [EMPTY\_ARM(LEFT)], Predicate [CLEAR([Block [name=D, weight=3]])], Predicate [HOLDING([Block [name=C, weight=2], RIGHT]), Predicate [HEAVIER([Block [name=D, weight=3], Block [name=C, weight=2]])]

For operator: [STACK(C, D, Arm: RIGHT)] to be used State needed to have: Predicate [ON([Block [name=C, weight=2], Block [name=D, weight=3]])]

-----

State:

Operators used: STACK(C, F, Arm: RIGHT)

Predicate [ON\_TABLE([Block [name=B, weight=2]])], Predicate [ON([Block [name=C, weight=2], Block [name=B, weight=2]])], Predicate [ON\_TABLE([Block [name=D, weight=3]])], Predicate [ON([Block [name=A, weight=1], Block [name=D, weight=3]])], Predicate [ON([Block [name=F, weight=1], Block [name=A, weight=1]])], Predicate [CLEAR([Block [name=F, weight=1]])], Predicate [EMPTY\_ARM(LEFT)], Predicate [CLEAR([Block [name=F, weight=1]])], Predicate [HOLDING([Block [name=C, weight=2], RIGHT]), Predicate [HEAVIER([Block [name=F, weight=1], Block [name=C, weight=2]])]

For operator: [STACK(C, F, Arm: RIGHT)] to be used State needed to have: Predicate [ON([Block [name=C, weight=2], Block [name=F, weight=1]])]

-----

State:

Operators used: UNSTACK(A, C, Arm: RIGHT)

Predicate [ON\_TABLE([Block [name=B, weight=2]])], Predicate [ON([Block [name=C, weight=2], Block [name=B, weight=2]])], Predicate [ON\_TABLE([Block [name=D, weight=3]])], Predicate [ON([Block [name=A, weight=1], Block [name=D, weight=3]])], Predicate [ON([Block [name=F, weight=1], Block [name=A, weight=1]])], Predicate [CLEAR([Block [name=F, weight=1]])], Predicate [EMPTY\_ARM(LEFT)], Predicate [EMPTY\_ARM(RIGHT)], Predicate [CLEAR([Block [name=A, weight=1]])], Predicate [ON([Block [name=A, weight=1], Block [name=C, weight=2]])]

Found contradicting predicates: [Predicate [CLEAR([Block [name=A, weight=1]])] with: [Predicate [ON([Block [name=F, weight=1], Block [name=A, weight=1]])]

-----

State:

**\*\*FILE SHORTENED\*\***

Figure 21: testing1.txt output file

```
<terminated> Main [Java Application] /usr/lib/jvm/java-8-oracle/bin/java (Nov 1, 2017)
Starting program
Please enter the file path to be read:
Filepath is: resources/testing1.txt
Map is: {GoalState=State [predicates=[Predicate [ON_TABLE([Block [name=B, weight=100, position=1],
Starting planner
Goal state is valid, calculating
Planning algorithm took 21494 milliseconds to find a plan
Output file is: par-activity-output1509568962433.txt
```

Figure 22: testing1.txt console output