Automated Warehouse Scenario: ASP Solver

Karan Dabas
Arizona State University
Tempe, Arizona, USA
kdabas1@asu.edu

Abstract

Given a warehouse with robots and various shelves with a variety of products on them, our object is to achieve complete automation such that robots could deliver the ordered products to the picking stations in the warehouse without any collision and in optimal possible time.

Introduction

I will be using Clingo answer set prograrmming language to develop the entire warehouse world and find the best possible plan of action to complete the orders in a given scenario. This project was one of the problem published in ASP Challenge 2019 submitted by Martin Gebser and Philipp Obermeier.

Problem Statement

The problem states that, we are given a set of robots in a warehouse. This warehouse is like one of the amazon fulfillment centers where robots automatically locate and bring the ordered products to the picking station to complete it. We are given an automated warehouse scenario where the warehouse is divided into a rectangular grid with multiple cells. A few robots that can move on the grid, such that they can only move horizontally or vertically from their current position on the grid. The robots cannot move diagonally , not allowed to swap positions with the adjacent robot and also cannot move out of the defined warehouse grid. Apart from the robots, the warehouse grid also have some shelves, each shelf contains a variety of products with some amount of units of such products. Some cells in the grid are marked as picking station, this is where the robot needs to bring the shelf with ordered products to complete a given order. Each order is mapped to one of the picking station. We are told that a robot can move under a shelf and pick it up and move the shelf to other cells such that no other shelf or robot is on that cell. No two robots can be on the same cell, and a robot carrying a shelf cannot move to a cell with other shelves on it. If such actions occur it will be treated as a collision. We are to avoid collision at all cost. The problem also states that some of the cells in the grid are defined as highway, this means that a robot with or without self can move freely on these cells with cannot drop off shelf on it at any given time. For an example setup of the warehouse see Figure-1. The objective is to complete the given orders with the minimum makespan. A makespan is defined as the greatest time step 't 'occurring within action of a plan. This means the solution should be such that it completes all orders in a scenario in the least possible time.

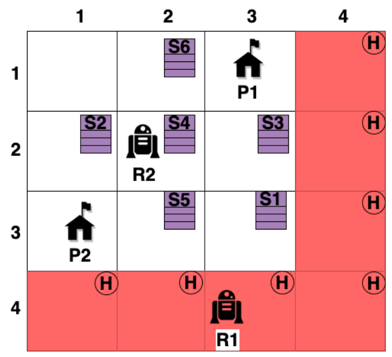


Figure 1: Warehouse setup from the inst1.lp. Here H represent the highway nodes, R represents robots, S represent shelves and P represent the picking stations.

Background Work

As seen from the problem statement, we have a variety of objects in a warehouse and a set action that can be performed. In order to identify these actions and the a way to represent such objects and respective actions of the warehouse world I first watched and understood the concepts taught to us in Module-3, Module-4 and Module-5 in class. The topics include basic arithmetic in clingo, clingo directives(#show, #include), pooling, negation, choice rules, how to use local and global variables, how to add constraints in clingo, cardinality bounds, concept of Generate-(Define)-Test to help define a world with a large number of actions and later based on the conditions prune out to find the most suitable solutions, representation of functions like one-to-one, onto, one-to-one-correspondence, how to do aggregation,summation in clingo(#count, #sum) along with how to do ASP optimization like #minimize, #maximize, how to identify and define states in a given system/world, define fluents, commonsense law of inertia, various state constraints, effect and precondition of actions, action constraints, and domain independent axioms like actions are exogenous, fluents are initially exogenous, uniqueness and existence of fluents.

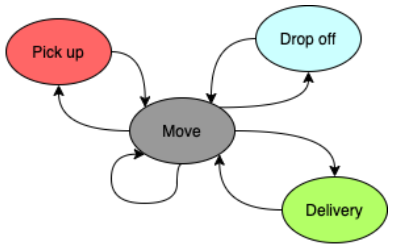


Figure 2: States the robot can go to in the defined automated warehouse world.

Based on the taught concepts, I was able identify and define the states of the robot in the warehouse problem. From Figure-2, the robot has 4 possible states that it can go to. A robot can **move** in the grid, it can **pickup** shelves, **drop off** shelves, **deliver** the shelf with ordered product to the picking station.

Methodology

After going through the problem statement, I decided to divide the entire warehouse problem into various subproblems. Beginning with defining all the rules based on the warehouse universe as stated in the problem description. Followed by the rules, I define my approach, which consists of setting up warehouse, defining functions and establishing the universe of warehouse. And lastly, implementing the optimization techniques used to filter out the solution that satisfies all of the above rules and requires the least amount of time.

Formulating Warehouse Rules

I start by defining the required rules that all the entities like robot, grid/cells in warehouse, highway cells, shelves and many other present in the warehouse need to follow. First rule, each robot may but not necessarily have to do any action(see Figure-2) given at any time instance. Another rule, a robot can only perform at most one action per time instance. This will restrict the robot to do only one transition state. A robot cannot swap locations among other robots. This will make sure that two robots in adjacent cells on the warehouse grid does not attempt to move onto each others cells. I defined a total of 30 such rules and a complete set of these rules can be found in the Appendix-A. Apart from this I also made sure invalid initial warehouse states are not allowed like there cannot be any holes in the warehouse grids, there cannot be any picking station or shelves on the defined highway cells on the grid.

Approach

The Clingo program that I developed to find the most suitable plan for robots to make successful delivery in the minimum time possible consists of multiple parts. Starting with setting up the warehouse, which takes in the initial state defined and setup the warehouse universe with all the mentioned entities and objects in the initial state file(given in inst1.lp,inst2.lp,...,inst5.lp). After the objects are added, I jump onto defining various functions that will help me keep track of the location of the said objects, their quantity(like product availability, number of shelves), length and width of the initiated warehouse grid and total number of cells on the grid. Once these functions are defined, I move onto establishing the warehouse universe, here I define all the state constraints based on the formulated rules from above sections, action effects, what happens to an entity at T+1 time instance if some action is performed on it at time T, sort and object declaration like that of robot movement, action constraints like a robot at most perform one action at a give time instance and domain independent axiom like fluents are initially exogenous, actions are exogenous, commonsense law of inertia that helps define what happens when an entity takes no action.

I organized my ASP solver such that, it first generates a search space of potential solutions in the warehouse universe, followed by defining some new atoms that can be used later and test phase that prune out solutions from the generate search space based on defined atoms that does not fit in the added constraints.

Setup Warehouse The given initial states of the warehouse is read and converted into atoms/functions of objects/entities present in the warehouse like number of cells in the warehouse grid, location of highway cells, robots, shelves, products, and location of products on the shelves, the location of the picking station where the order delivery is to be made, the orders are also defined in this part which consists of a set of products asked to be delivered at a specific picking station. Apart from the mentioned setup I have also added some checks on the initial state for example, an initial state cannot have a shelf located on a defined highway cell or a picking station cannot be on a defined highway cell. If such checks are not met the ASP will return unsatisfiable model. Please see Appendix-C for the Clingo code under the name ”setupwarehouse.lp”.

Defining Functions In order to keep track of the quantity of above defined atoms in the warehouse I define functions that uses count aggregate in Clingo #count. Here I also define functions that return the total number of cells in the warehouse from the initial state, functions for width and height of the grid. Given the total number of cells defined and width and height of the grid I added a constraint that keeps a check for holes on the grid. This check is crucial as mentioned in the problem description a grid cannot have any holes in it. Please see Appendix-C for the Clingo code under the name ”gridcontrol.lp”.

Establishing Complete Universe Once we have our warehouse setup and basic functions defined I write down the main compute ASP file that finds all possible solutions for completing the orders using the robots in the warehouse while still keeping out a check on the formulated rules mentioned above. In here, I start with defining robot movements under object declaration. A robot can only move vertically or horizontally adjacent cells. This means its movement can only be one of the 4 options (1,0), (0,1), (-1,0), (0,-1). This can be found in robotmovement.lp file in Appendix-C. The main compute ASP starts with establishing **generate** space where each robot may but not necessarily have to do any action(move,pickup,dropoff,deliver) in a given time instance. This was achieved by using choice rules. The actions robotmove, deliver, dropoff, pickup as used in generate space is also used to **define** a global action called occurs that keeps track of these individual actions. This occurs action can be used to find out the total number of actions performed in a particular solution. I later went on to make **test** phase that uses the listed rules to construct various action and state constraints to prune out the best solution out of the stack. Please see Appendix-A for the complete list of rules. The code for this section can be found in Appendix-C under the name ”compute.lp”. I define the goal of completing the deliveries as constraint where the delivery of ordered product I must be equal to 0 products at the finish time t and a U amount of product at the start t=0 in a given order. This means that at the finish time t all the orders are fulfilled(product asked turn to 0 in a given order).

Optimization Technique

As stated in the problem statement, the complete plan of actions of completing all the orders in a given initial state must be done in a minimum makespan. This means that the ASP must return the most optimal plan of actions that require the least amount of time to complete all the orders. To make this possible I define a function that return the total time take by the solution to fulfill orders called overallTime(t), where t is the total time taken. This t is calculated by doing a count aggregate on the occurs fuction w.r.t time. This means counting out the unique time instance in the occurs functions. On this ovverallTime function I do ASP optimization called #minimium{T:overallTime(T)}. This helps ASP to find out the most optimal solution that has the least makespan. Please see Appendix-C last few lines in ”compute.lp”

Results and Analysis

From ”compute.lp” file in Appendix-C you can see that I am returning the most optimal solution that has the least time along with other information like overallTime that tell the total time take by a given plan of actions and totalActions that return the total number of ”occurs” actions needed in the plan to fulfill all the orders in the instance file. See Table-1 that shows the solutions found my by ASP solver for the given instance files in terms of overallTime and totalActions.

Inst	Input Query	oveallTime	totalAction
1	clingo compute.lp inst1.lp -c t=15	t=13	24
2	clingo compute.lp inst2.lp -c t=15	t=11	22
3	clingo compute.lp inst3.lp -c t=15	t=7	12
4	clingo compute.lp inst4.lp -c t=15	t=10	18
5	clingo compute.lp inst5.lp -c t=15	t=6	10

Table 1: Input Query and corresponding optimal solution provided by my ASP program with respect to the given instance files.

As you see, my ASP solver found a solution for inst1.lp with overallTime=13 and totalActions performed in the solution is equal to 24. For inst2.lp I am getting a solution in 11 time steps with total actions equal to 22. See Table-1 for complete list. Below I have two output solutions for inst3.lp and inst5.lp as an example. The rest of the outputs can be found in Appendix-B.

```
############Instance -3###########

occurs(object(robot,1),move(-1,0),0)
occurs(object(robot,2),move(1,0),0)
occurs(object(robot,1),move(-1,0),1)
occurs(object(robot,2),pickup,1)
occurs(object(robot,1),move(0,-1),2)
occurs(object(robot,2),move(0,-1),2)
occurs(object(robot,1),pickup,3)
occurs(object(robot,2),deliver(2,4,1),3)
occurs(object(robot,1),move(1,0),4)
occurs(object(robot,2),move(1,0),4)
occurs(object(robot,1),move(0,-1),5)
occurs(object(robot,1),deliver(1,2,1),6)

overallTime(7)
totalActions(12)

OPTIMUM FOUND

Models       : 5
  Optimum    : yes
Optimization : 7
CPU Time     : 23.823s
```

Listing 1: Optimal solution found by my ASP program for inst3.lp

From Table-1 you can see that the more the number of actions the greater the overall time to complete the orders. This is because all the instance files are mapped with a 4x4 warehouse grid and all of them have 2 robots to do work. The only things that is different among the files are the number of shelves and products, initial location of robots and shelves, and the number of picking station. If the number of robots are increased then the overall time might decrease even further as then there will be more robots to work in parallel.

Another interesting thing that can be noticed from the outputs shown above and in Appendix-B is that the CPU time to find the optimal solution is increased when the solution has more ”occurs” actions. In inst1.lp the totalActions equals 24 and the CPU time to find it is 72.216s which more when compared with the rest inst2 takes 30.180s with total actions equal to 22, inst3 takes 23.8s and total actions of 12, inst4 takes 68.3 and total actions of 18 and inst5 takes CPU time of 20.95s and total actions equal to 10.

```
#############Instance -5###########

occurs(object(robot,1),move(-1,0),0)
occurs(object(robot,2),move(-1,0),0)
occurs(object(robot,1),move(-1,0),1)
occurs(object(robot,2),pickup,1)
occurs(object(robot,2),move(0,1),2)
occurs(object(robot,1),pickup,3)
occurs(object(robot,2),deliver(1,3,4),3)
occurs(object(robot,1),move(-1,0),4)
occurs(object(robot,2),move(0,-1),4)
occurs(object(robot,1),deliver(1,1,1),5)

overallTime(6)
totalActions(10)
Optimization: 6
OPTIMUM FOUND

Models        : 6
  Optimum     : yes
Optimization  : 6
CPU Time      : 20.956s
```

Listing 2: Optimal solution found by my ASP program for inst5.lp

Conclusion

I believe that with the help of knowledge gained by the lessons taught in the course CSE 579: Knowledge representation and reasoning, and various programming assignment helped me complete this project in its entirety.leave no stone unturned, a complete overview of my work can be viewed in appendix section where I have attached all my rough work that helped me sketch out a perfect solution for the given problem. I have applied multiple techniques that were taught during lecturers like concept of generate-define-test, ASP aggregates, ASP optimization, transition system in ASP, representing action domain in ASP and many more. All of this helped me get the desired solution for the given inst files. Since I don't have anything to compare my results with other than an example solution given for inst1.lp in problem description to which my programmed ASP also gave a similar solution with the same time and actions taken. My program is more generic can accommodate for any number of robots, shelves, products or grid size of the warehouse.

Future Work

As for the scope of the given warehouse problem description, I believe my codebase meets all the expectations and provides optimal solutions with various fail safe constraints in place to prevent wrong input data/initial state. I think that this logic based solution can further be extended into a more user-friendly application that could be used in real-world scenarios. Maybe using clingo module in python along with python based web-based monitoring and recommendation system that keeps track of various robots in a warehouse and provides recommendations for the most optimal plan to get a product to the delivery stations. Such system can also be used in figuring out a most optimal strategy for facility expansion, like with increase number of robots what will be the expected decrease in operation time and overall cost of management.

Appendix-A

Below is the complete rule book that I developed based on my understating of the given automated warehouse problem.

1. Each Robot may but not necessarily have to do any action(move,pickup,dropoff) in a given time instance.
2. A robot can only perform at most one action per time instance.
3. A robot cannot swap with other robots on it.
4. A robot cannot move outside the initialized grid in Warehouse.
5. A same robot cannot be on more than one cell/location.
6. At most one robot on a given cell/location.
7. A robot cannot go to a cell with shelf if it has a shelf on it before.
8. A shelf can be at only one cell/location. (only on one node/robot in a grid and can only be either on a node or on a robot)
9. A Shelf cannot be at two cells at a given time instance T.
10. A Shelf cannot be at two robots at a given time instance T.
11. A Shelf cannot be at at a cell and on robot at a given time instance T.
12. Two different shelves cannot be on the same robot at a given time instance T.
13. Two or more shelves cannot be on one node.
14. A robot can only pickup one shelf which is on its current cell/location.
15. Only one robot can pickup the shelf.
16. Only the shelf picked up by robot should be on it.
17. A robot cannot pick other shelf if already has one on it.
18. A robot cannot dropoff shelf if it does not have it.
19. A robot cannot put down shelf on highway
20. Shelf cannot be on the Highway.
21. picking station cannot be on the Highway.
22. A robot need to be on the order picking station.
23. Delivery is only possible if robot has the product on its shelf that was ordered.
24. Delivery is only possible if robot has the shelf with the ordered product.
25. Delivery not allowed if the asked amount of units of a product is more than available units of the product on the shelf.

26. A robot does not change location if no move action is done on it.
27. A shelf remains at the same cell/location if no action(like pickup) is done on it.
28. A shelf remain on a robot if no dropoff action is done on it.
29. An order is not complete if it is not yet delivered.
30. Units of product on a shelf remains the same if not delivered in any order.

Appendix-B

Output of my ASP solver for the rest of the instance files provided with the problem description.

```
#############Instance -1###########

occurs(object(robot,1),move(-1,0),0)
occurs(object(robot,2),move(-1,0),0)
occurs(object(robot,1),move(-1,0),1)
occurs(object(robot,2),pickup,1)
occurs(object(robot,1),pickup,2)
occurs(object(robot,2),move(0,1),2)
occurs(object(robot,2),deliver(1,3,4),3)
occurs(object(robot,1),move(-1,0),4)
occurs(object(robot,2),move(0,-1),4)
occurs(object(robot,1),deliver(1,1,1),5)
occurs(object(robot,2),putdown,5)
occurs(object(robot,1),putdown,6)
occurs(object(robot,2),move(1,0),6)
occurs(object(robot,1),move(1,0),7)
occurs(object(robot,2),move(1,0),7)
occurs(object(robot,2),pickup,8)
occurs(object(robot,1),move(0,-1),8)
occurs(object(robot,1),pickup,9)
occurs(object(robot,2),move(0,-1),9)
occurs(object(robot,1),move(1,0),10)
occurs(object(robot,2),deliver(3,4,1),10)
occurs(object(robot,1),move(0,-1),11)
occurs(object(robot,2),move(1,0),11)
occurs(object(robot,1),deliver(2,2,1),12)

overallTime(13)
totalActions(24)
Optimization: 13
OPTIMUM FOUND

Models       : 3
  Optimum    : yes
Optimization : 13
CPU Time     : 72.216s
```

Listing 3: Optimal solution found by my ASP program for inst1.lp

```
#############Instance -2###########

occurs(object(robot,1),move(-1,0),0)
occurs(object(robot,2),move(0,1),0)
occurs(object(robot,1),move(0,-1),1)
occurs(object(robot,2),pickup,1)
occurs(object(robot,1),pickup,2)
occurs(object(robot,2),move(-1,0),2)
occurs(object(robot,1),move(1,0),3)
occurs(object(robot,2),deliver(1,1,1),3)
occurs(object(robot,1),move(0,1),4)
occurs(object(robot,2),move(0,-1),4)
occurs(object(robot,1),move(0,1),5)
occurs(object(robot,2),putdown,5)
occurs(object(robot,1),move(-1,0),6)
occurs(object(robot,2),move(1,0),6)
occurs(object(robot,1),move(-1,0),7)
occurs(object(robot,2),pickup,7)
occurs(object(robot,1),move(-1,0),8)
occurs(object(robot,2),move(1,0),8)
occurs(object(robot,1),move(0,-1),9)
occurs(object(robot,2),move(0,-1),9)
occurs(object(robot,1),deliver(1,3,2),10)
occurs(object(robot,2),deliver(2,2,1),10)

overallTime(11)
totalActions(22)

Optimization: 11
OPTIMUM FOUND

Models       : 4
  Optimum    : yes
Optimization : 11
CPU Time     : 30.180s
```

Listing 4: Optimal solution found by my ASP program for inst2.lp

```
#############Instance -4###########

occurs(object(robot,1),move(0,-1),0)
occurs(object(robot,2),move(0,-1),0)
occurs(object(robot,1),move(-1,0),1)
occurs(object(robot,2),pickup,1)
occurs(object(robot,1),move(-1,0),2)
occurs(object(robot,2),move(-1,0),2)
occurs(object(robot,1),pickup,3)
occurs(object(robot,2),putdown,3)
occurs(object(robot,1),move(0,-1),4)
occurs(object(robot,2),move(0,1),4)
occurs(object(robot,1),move(1,0),5)
occurs(object(robot,2),move(1,0),5)
occurs(object(robot,2),move(0,1),6)
occurs(object(robot,2),pickup,7)
occurs(object(robot,1),deliver(2,2,1),8)
occurs(object(robot,2),move(-1,0),8)
occurs(object(robot,1),deliver(3,2,2),9)
occurs(object(robot,2),deliver(1,1,1),9)

overallTime(10)
totalActions(18)
Optimization: 10
OPTIMUM FOUND

Models       : 5
  Optimum    : yes
Optimization : 10
CPU Time     : 68.350s
```

Listing 5: Optimal solution found by my ASP program for inst4.lp

Appendix-C

My code for robot movement and warehouse setup that takes all the input from the given initial state and defines all necessary functions, my code for sort and object declaration, actions are exogenous in the warehouse, common sense law of inertia, and state constraints.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% setupwarehouse.lp %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
pair(X,Y):- init(object(node,C),value(at,pair(X,Y))).
nodeAt(C,pair(X,Y)):-init(object(node,C),value(at,pair(X,Y))).
node(C):- nodeAt(C,pair(X,Y)).

highwayAt(C,pair(X,Y)):- init(object(highway,C),value
(at,pair(X,Y))).
highway(C):- highwayAt(C,pair(X,Y)).

robotAt(R,object(node,C),0):- init(object(robot,R),value
(at,pair(X,Y))), nodeAt(C,pair(X,Y)).
robot(R):- robotAt(R,object(node,C),0).

shelfAt(S,object(node,C),0):- init(object(shelf,S),value
(at,pair(X,Y))), nodeAt(C,pair(X,Y)), not highwayAt
(C,pair(X,Y)).
shelf(S):- shelfAt(S,object(node,C),0).

productOn(I,object(shelf,S),with(quant,U),0):- init(object
(product,I),value(on,pair(S,U))).

product(I):- productOn(I,object(shelf,S),with(quant,U),0).

pickingStationAt(P,C):- init(object(pickingStation,P),value
(at,pair(X,Y))), not highwayAt(C,pair(X,Y)), nodeAt(C,pair
(X,Y)).
pickingStation(P):- pickingStationAt(P,C).

order(O):- init(object(order,O),value(pickingStation,P)).
deliverAt(O,object(node,C),contains(I,U),0):- init(
object(order,O),value(line,pair(I,U))), pickingStationAt
(P,C), init(object(order,O),value(pickingStation,P)).
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% gridcontrol.lp %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Length and width of the Warehouse grid.
numGridWidth(N):- N=#count{X:nodeAt(C,pair(X,Y))}.
numGridLength(N):- N=#count{Y:nodeAt(C,pair(X,Y))}.
% Total number of cells/nodes initialized during the init state.
numTotalCells(N):- N=#count{C:node(C)}.

% Number of robots in the Warehouse.
numRobot(N):- N=#count{R:robot(R)}.

% Number of Shelves in the Warehouse.
numShelves(N):- N=#count{S:shelf(S)}.

% number of Orders
numOrders(N):- N=#count{O:order(O)}.

% Number of Picking Stations in Warehouse.
numPickingStation(N):- N=#count{P:pickingStation(P)}.

% Number of Highway cells declaredin a Warehouse.
numHighway(N):- N=#count{C:highway(C)}.

% Number of products initially declared in a Warehouse.
numProducts(N):- N=#count{I:product(I)}.

%Check Grid for holes, if found the initial states are not valid
:- numTotalCells(N),numGridWidth(W),numGridLength(L), not N = W*L.


#show numGridWidth/1.
#show numGridLength/1.
#show numTotalCells/1.
#show numRobot/1.
#show numShelves/1.
#show numOrders/1.
#show numPickingStation/1.
#show numHighway/1.
#show numProducts/1.
```

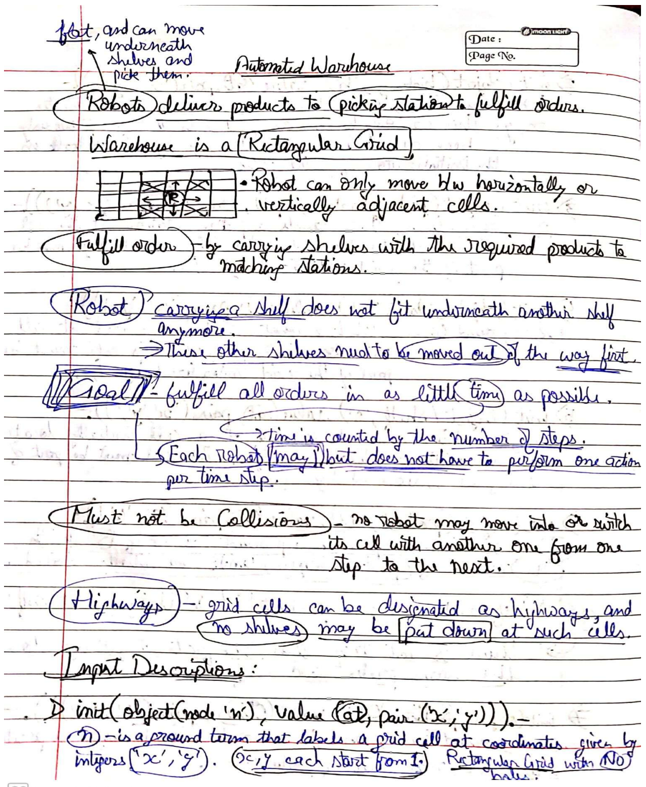
```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% robotmovement.lp %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Only horizontal and vertical movement
possible for a robot.
move(1,0;0,1;-1,0;0,-1).
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% compute.lp %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
#include "gridControl.lp".
#include "robotMovement.lp".

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Generate Space %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Each Robot may but not necessarily have to do any action(move,
pickup,dropoff,deliver) in a given time instance.
{robotmove(R,DX,DY,T): move(DX,DY)}1:-  robot(R), T=0..t-1.
{pickup(R,S,T): shelf(S)}1:- robot(R), T=0..t-1.
{dropoff(R,S,T):shelf(S)}1:- robot(R), T=0..t-1.
{delivery(R,O,S,I,UI,T):deliverAt(O,object(node,C),
contains(I,U2),T), productOn(I,object(shelf,S),with
(quant,U),T), UI=1..U}1:- robot(R), T=0..t-1.


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Define Occurs %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
occurs(object(robot,R),move(DX,DY),T):- robotmove(R,DX,DY,T).
occurs(object(robot,R),pickup,T):- pickup(R,S,T).
occurs(object(robot,R),putdown,T):- dropoff(R,S,T).
occurs(object(robot,R),deliver(O,I,U),T):- delivery(R,O,S,I,U,T).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Constraints %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%% Robot Constraints %%%%%
% A robot can only perform atmost one action per time instance.
:- occurs(object(robot,R),Ac1,T), occurs(object(robot,R),Ac2,T),
Ac1!=Ac2, T=0..t-1.

% A robot cannot swap with other robots on it.
:- occurs(object(robot,R1),move(DX1,DY1),T), robotAt(R1,
object(node,C1),T), occurs(object(robot,R2),move(DX2,DY2),T),
robotAt(R2,object(node,C2),T), R1!=R2, C1!=C2, nodeAt(C1,
pair(X1,Y1)),nodeAt(C2,pair(X2,Y2)), X1+DX1 = X2,
X2+DX2 = X1, Y1+DY1=Y2, Y2+DY2=Y1.

% A robot cannot move outside the initialized grid in Warehouse.
:- robotmove(R,DX,DY,T), robotAt(R,object(node,C),T), nodeAt(C,
pair(X,Y)), X+DX > L, numGridLength(L).
:- robotmove(R,DX,DY,T), robotAt(R,object(node,C),T), nodeAt(C,
pair(X,Y)), X+DX < 1.
:- robotmove(R,DX,DY,T), robotAt(R,object(node,C),T), nodeAt(C,
pair(X,Y)), Y+DY > W, numGridWidth(W).
:- robotmove(R,DX,DY,T), robotAt(R,object(node,C),T), nodeAt(C,
pair(X,Y)), Y+DY < 1.

% A same robot cannot be on more than one cell/location.
:- robotAt(R,object(node,C1),T), robotAt(R,object(node,C2),T),
C1!=C2.
% Atmost one robot on a given cell/location.
:- robotAt(R1,object(node,C1),T), robotAt(R2,object(node,C1),T),
R1!=R2.
% A robot cannot go to a cell with shelf if it has a shelf on
it before.
:- shelfAt(S,object(robot,R),T), robotAt(R,object(node,C),T),
shelfAt(S1,object(node,C),T), S!=S1.

%%%%% Shelf Constraints %%%%%
% A shelf can be at only one cell/location. (only on one node/robot
in a grid and can only be either on a node or on a robot)
% A Shelf cannot be at two cells at a given time instance T.
:- shelfAt(S,object(node,C),T), shelfAt(S,object(node,C1),T),
C!=C1, T=0..t-1.
% A Shelf cannot be at two robots at a given time instance T.
:- shelfAt(S,object(robot,R),T), shelfAt(S,object(robot,R1),T),
R!=R1, T=0..t-1.
% A Shelf cannot be at at a cell and on robot at a given
time instance T.
:- shelfAt(S,object(node,C),T), shelfAt(S,object(robot,R),T),
T=0..t-1.
% Two different shelves cannot be on the same robot at a given
time instance T.
:- shelfAt(S1, object(robot,R),T), shelfAt(S2,object(robot,R),T),
S1!=S2, T=0..t-1.
% Two or more shelves cannot be on one node.
:- not {shelfAt(S,object(node,C),T): shelf(S)}1, node(C),
T=0..t-1.

%%%%% Pickup Constraints %%%%%
% A robot can only pickup one shelf which is on its current
cell/location.
```

```
:- robotAt(R, object(node,C),T), shelfAt(S, object(node,C1),T),
pickup(R,S,T), C!=C1, T=0..t-1.

% Only one robot can pickup the shelf.
:- pickup(R,S,T), pickup(R1,S,T), R!=R1.

% Only the shelf picked up by robot should be on it.
:- pickup(R,S,T), shelfAt(S,object(robot,R1),T), R!=R1, T=0..t-1.

% A robot cannot pick other shelf if already has one on it.
:- shelfAt(S,object(robot,R),T), occurs(object(robot,R),pickup,T),
pickup(R,S1,T). %Check on this.
%:- pickup(R,S1,T), shelfAt(S,object(robot,R),T), S!=S1.

%%%%% Dropoff Constraints %%%%%
% A robot cannot dropoff shelf if it does not have it.
:- dropoff(R,S,T), not shelfAt(S,object(robot,R),T), T=0..t-1.
% A robot cannot put down shelf on highway
:- shelfAt(S,object(robot,R),T), dropoff(R,S,T),
robotAt(R,object(node,C),T), highway(C).
%% A shelf cant be putDown by 2 robots
%:- 2{putdown(R,S,T): robot(R)}, shelf(S).

%%%%% Highway Constraints %%%%%
% Shelf cannot be on the Highway.
:- highway(C), shelfAt(S,object(node,C),T), T=0..t-1.
% picking station cannot be on the Highway.
% already added in the setupWarehouse.lp

%%%%% Delivery Constraints %%%%%
% A robot need to be on the order picking station.
:- delivery(R,O,S,I,U,T), robotAt(R,object(node,C),T),
deliverAt(O,object(node,C1),contains(I,U),T), C!=C1.
% Delivery is only possible if robot has the product
on its shelf that was ordered.
:- shelfAt(S,object(robot,R),T), productOn(I,object(shelf,S),
with(quant,0),T), delivery(R,O,S,I,_,T).
% Delivery is only possible if robot has the shelf with the
ordered product.
:- not shelfAt(S,object(robot,R),T), productOn(I,object(shelf,S),
with(quant,_),T), delivery(R,O,S,I,_,T).
% Delivery of more than the asked units of products in an order
is not possible.
:- delivery(R,O,S,I,U1,T), deliverAt(O,object(node,C),
contains(I,U2),T), U1>U2.

% Delivery not allowed if the asked amount of units of a product
is more than available units of the product on the shelf.
:- delivery(R,O,S,I,U1,T), productOn(I,object(shelf,S),
with(quant,U2),T), U1>U2.
:- deliverAt(O,object(node,C),contains(I,U),T), U<0.
:- productOn(I,object(shelf,S),with(quant,U),T), U<0.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Action Effect %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Moving a robot.
robotAt(R,object(node,C),T+1):- robotAt(R,object(node,C1),T),
robotmove(R,DX,DY,T), nodeAt(C1, pair(X,Y)), nodeAt(C,
pair(X+DX,Y+DY)).
% Robot picking a shelf.
shelfAt(S,object(robot,R),T+1):- robot(R), robotAt(R,object(
node,C),T), shelfAt(S,object(node,C),T), pickup(R,S,T).
% Robot dropoff shelf.
shelfAt(S,object(node,C),T+1):- robotAt(R,object(node,C),T),
shelfAt(S,object(robot,R),T), dropoff(R,S,T).
% Robot delivering the order, so the amount of item required
get reduced by the amount delivered and some for product
inventory in Warehouse.
deliverAt(O,object(node,C),contains(I,U-U1),T+1):-
deliverAt(O,object(node,C),contains(I,U),T),delivery(R,O,S,I,U1,T).
productOn(I,object(shelf,S),with(quant,U-U1),T+1):-
productOn(I,object(shelf,S),with(quant,U),T),delivery(R,O,S,I,U1,T).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Law of Inertia %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% A robot does not change location if no move action is done on it.
robotAt(R,object(node,C),T+1):- robotAt(R,object(node,C),T),
not robotmove(R,_,_,T), T<t.
% A shelf remains at the same cell/location if no action
(like pickup) is done on it.
shelfAt(S,object(node,C),T+1):- shelfAt(S,object(node,C),T),
not pickup(_,S,T), T<t.
% A shelf remain on a robot if no dropoff action is done on it.
shelfAt(S,object(robot,R),T+1):- shelfAt(S,object(robot,R),T),
not dropoff(R,S,T), T<t.
% An order is not complete if it is not yet delivered.
deliverAt(O,object(node,C),contains(I,U),T+1):-
deliverAt(O,object(node,C),contains(I,U),T),
productOn(I,object(shelf,S),with(quant,U1),T),
not delivery(_,O,S,I,U1,T), T<t.
% Units of product on a shel remains the same if
not delivered in any order.
productOn(I,object(shelf,S),with(quant,U),T+1):-
productOn(I,object(shelf,S),with(quant,U),T),
not delivery(_,_,S,I,_,T), T<t.


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Goal State %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
:- not deliverAt(O,object(node,C),contains(I,0),t),
deliverAt(O,object(node,C),contains(I,U),0).

#show occurs/3.


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Find the minimum %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Total number of actions in a plan.
totalActions(N):- N=#count{Obj,Ac,T:occurs(Obj,Ac,T)}.
% We want to minimize the Makespan(Time t for completing
the action plan).

overallTime(TT):- TT=#count{T:occurs(Obj,Ac,T)}.
#minimize{TT:overallTime(TT)}.
%#minimize{N:totalActions(N)}.
#show totalActions/1.
#show overallTime/1.
```

Appendix-D

My rough work on sketching out the tasks from the entire problem description. See Figure- 3,4,5,6,7



Date :
Page No.

flat, and can move underneath shelves and pick them.

Automated Warehouse

Robots deliver products to picking stations to fulfill orders.

Warehouse is a Rectangular Grid

• Robot can only move b/w horizontally or vertically adjacent cells.

Fulfill order - by carrying shelves with the required products to matching stations.

Robot carrying a shelf does not fit underneath another shelf anymore.
⇒ These other shelves need to be moved out of the way first.

Goal - fulfill all orders in as little time as possible.
→ time is counted by the number of steps.
→ Each robot (may) but does not have to perform one action per time step.

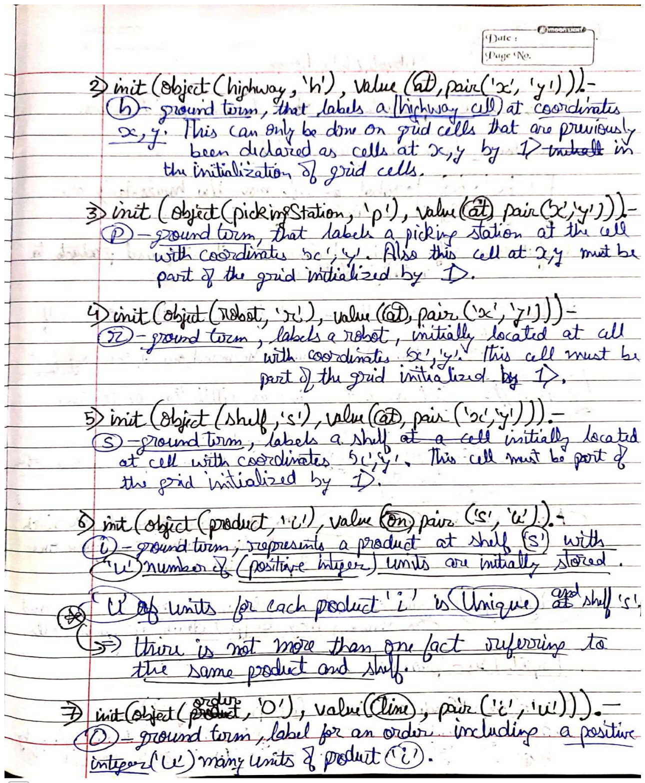
Must not be Collisions - no robot may move into or switch its cell with another one from one step to the next.

Highways - grid cells can be designated as highways, and no shelves may be put down at such cells.

Input Descriptions:

1) init(object(node 'n'), value(at, pair('x','y'))).-
n - is a ground term that labels a grid cell at coordinates given by integers ('x','y'). x, y each start from 1. Rectangular Grid with NO holes.

Figure 3: Rough work 1.


Date :
Page No.

2) init(object(highway, 'h'), value(at, pair('x', 'y'))).-
h - ground term, that labels a highway cell at coordinates x, y. This can only be done on grid cells that are previously been declared as cells at x, y by 1) ~~[illegible]~~ in the initialization of grid cells.

3) init(object(pickingStation, 'p'), value(at, pair('x','y'))).-
p - ground term, that labels a picking station at the cell with coordinates 'x', 'y'. Also this cell at x, y must be part of the grid initialized by 1).

4) init(object(robot, 'r'), value(at, pair('x', 'y'))).-
r - ground term, labels a robot, initially located at cell with coordinates 'x', 'y'. This cell must be part of the grid initialized by 1).

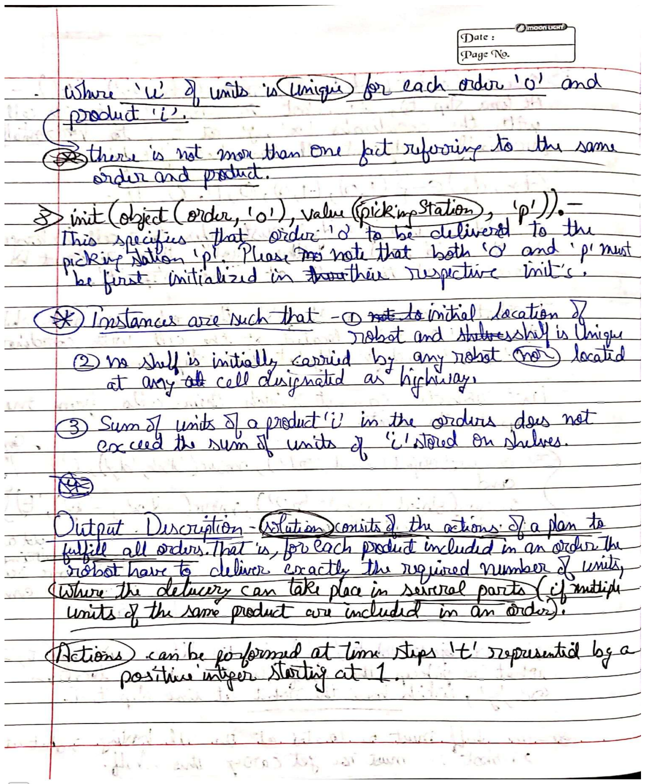
5) init(object(shelf, 's'), value(at, pair('x','y'))).-
s - ground term, labels a shelf ~~at a cell~~ initially located at cell with coordinates 'x','y'. This cell must be part of the grid initialized by 1).

6) init(object(product, 'i'), value(on, pair('s', 'u'))).-
i - ground term, represents a product at shelf 's' with 'u' number of (positive integer) units are initially stored.

* 'u' ~~of~~ of units for each product 'i' is Unique and shelf 's'.
⇒ there is not more than one fact referring to the same product and shelf.

7) init(object(~~product~~ order, 'O'), value(line, pair('i', 'u'))).-
O - ground term, label for an order including a positive integer ('u') many units of product 'i'.

Figure 4: Rough work 2.


Date :
Page No.

Where 'u' of units is Unique for each order 'o' and product 'i'.

* there is not more than one fact referring to the same order and product.

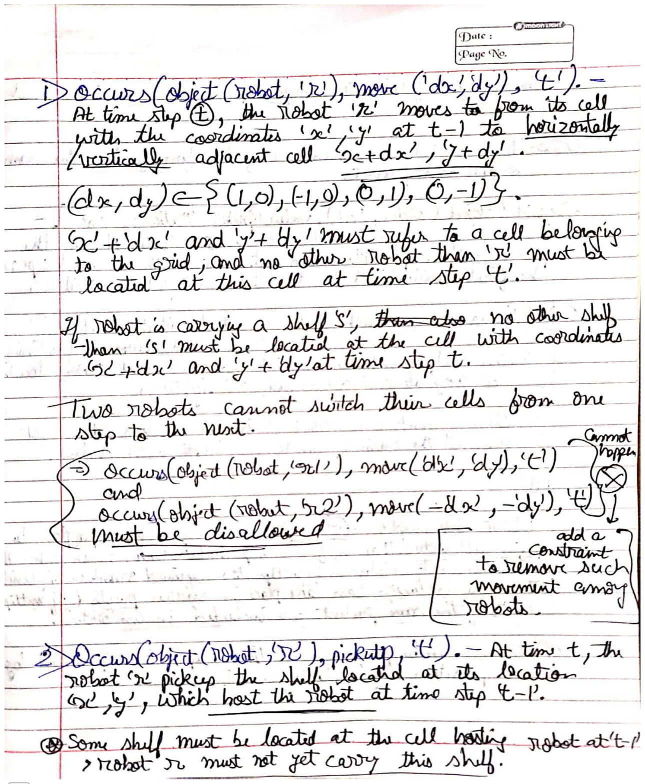
8) init(object(order, 'o'), value(pickingStation, 'p')).-
This specifies that order 'o' to be delivered to the picking station 'p'. Please ~~no~~ note that both 'o' and 'p' must be first initialized in ~~[illegible]~~ their respective init's.

* Instances are such that - ① ~~not to~~ initial location of robot and ~~shelves~~ shelf is Unique
② no shelf is initially carried by any robot nor located at any ~~all~~ cell designated as highway.
③ Sum of units of a product 'i' in the orders does not exceed the sum of units of 'i' stored on shelves.
~~④~~

Output Description - solution consists of the actions of a plan to fulfill all orders. That is, for each product included in an order the robot have to deliver exactly the required number of units, (where the delivery can take place in several parts (if multiple units of the same product are included in an order).

Actions can be performed at time steps 't' represented by a positive integer starting at 1.

Figure 5: Rough work 3.


Date :
Page No.

1) Occurs(object(robot, 'r'), move('dx','dy'), 't').-
At time step t, the robot 'r' moves ~~to~~ from its cell with the coordinates 'x', 'y' at t-1 to horizontally / vertically adjacent cell 'x+dx', 'y+dy'.
$(dx, dy) \in \{(1,0), (-1,0), (0,1), (0,-1)\}$.

'x' + 'dx' and 'y' + 'dy' must refer to a cell belonging to the grid, and no other robot than 'r' must be located at this cell at time step 't'.

If robot is carrying a shelf 's', ~~than also~~ no other shelf than 's' must be located at the cell with coordinates 'x'+'dx' and 'y'+'dy' at time step t.

Two robots cannot switch their cells from one step to the next.
⇒ Occurs(object(robot, 'r1'), move('dx', 'dy'), 't')
and
occurs(object(robot, 'r2'), move(-'dx', -'dy'), 't')
must be disallowed

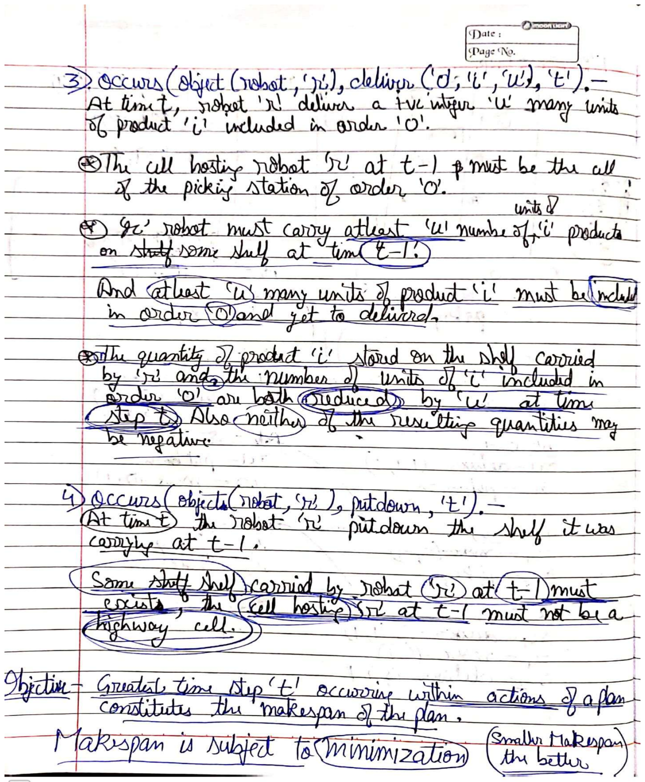
Cannot happen

add a constraint to remove such movement among robots.

2) Occurs(object(robot, 'r'), pickup, 't').- At time t, the robot 'r' pickup the shelf located at its location 'x','y', which host the robot at time step 't-1'.

* Some shelf must be located at the cell hosting robot at 't-1', robot r must not yet carry this shelf.

Figure 6: Rough work 4.


Date :
Page No.

3) Occurs(object(robot, 'r'), deliver('o', 'i', 'u'), 't').-
At time t, robot 'r' deliver a +ve integer 'u' many units of product 'i' included in order 'O'.

* The cell hosting robot 'r' at t-1 ~~p~~ must be the cell of the picking station of order 'O'.

* 'r' robot must carry atleast 'u' number of units of 'i' products on ~~shelf~~ some shelf at time t-1.

And atleast 'u' many units of product 'i' must be included in order 'O' and yet to delivered.

* The quantity of product 'i' stored on the shelf carried by 'r' and the number of units of 'i' included in order 'O' are both reduced by 'u' at time step t. Also neither of the resulting quantities may be negative.

4) Occurs(objects(robot, 'r'), putdown, 't').-
At time t the robot 'r' putdown the shelf it was carrying at t-1.

Some ~~shelf~~ shelf carried by robot 'r' at t-1 must exists, the cell hosting 'r' at t-1 must not be a highway cell.

Objective - Greatest time step 't' occurring within actions of a plan constitutes the makespan of the plan.
Makespan is subject to minimization (Smaller Makespan the better)

Figure 7: Rough work 5.

Appendix-E

See Table 2 where I have listed breakup of all the tasks I completed along with the time series for this project.

EDOC	Tasks	Status
2/28	Identify all objects in Warehouse	Done
3/01	Complete code to setup warehouse with given initial states	Done
3/01	Define robot movement	Done
3/05	Define grid control	Done
3/06	Identify all actions in a warehouse	Done
3/10	Code for actions are exogenous	Done
3/12	Define common sense law of inertia for warehouse world	Done
3/15	Code for common sense law of inertia	Done
3/20	Identify all required constraints (Robot move, pickup, dropoff, shelf, picking station, delivery, highway	Done
3/22	Code for robot movement constraints	Done
3/24	Code for shelf constraints	Done
3/28	Code for robot pickup action constraint on shelf	
3/30	Code for robot dropoff constraint on shelf	Done
3/31	Code for highway constraint	Done
4/02	Code for picking station constraint	Done
4/05	Code for delivery constraint	Done
4/05	Add optimization to find the minimum makespan	Done
4/05	Deliver with final results	Done

Table 2: List of tasks and current progress of the project. EDOC represent Expected date of completion.

All the challenges I faced till date while completing this project:

1. Identify all the object in the problem.
2. Understand the complex input description given in problem.
3. How to represent the warehouse initial state in clingo.
4. Figure out all the constraints mentioned across the problem.
5. Understand the complex output description given in problem.
6. List down all the actions allowed in the problem.
7. How to represent all actions in clingo.
8. Figure out how to define the effects and preconditions of actions.
9. How to define common sense law of inertia in clingo for this problem.
10. Identify all possible collisions among robots, robot with shelf on it and other shelves.
11. Define the action over time such that it could later be used to minimize the overall time for optimal solution(minimum makespan).