Automated Warehouse Scenario

Individual Research Report

Abhijit Chakraborty

Email: achakr40@asu.edu

Abstract

As part of the Knowledge Representation and Reasoning individual project that I just fin-ished, I'll share my final conclusions and results in this report. I selected the Automated Warehouse Scenario from the list of options available for my research study. This report will include the following topics: an over-view of the overall problem and its subproblems, my strategy for addressing the problem and its subproblems, an analysis of the key results I obtained, the project's overall conclusion, and how I intend to apply what I've learned here in my professional life. I hope you find this report helpful.

Defining the Project's Problem

I chose the Automated Warehouse Scenario for my Knowledge Representation and Reasoning individual project. The major project goal in this case is to show that I can use KR technologies to do auto-mated reasoning about actions. Robots transport items to picking stations in an automated warehouse to fulfill or-ders. To complete these orders, robots must transport shelves with the needed items to the appropriate picking stations. The problem statement, or overarching objective, is to complete all orders in the shortest amount of time fea-sible, The problem statement, or overarching objective, is to complete all orders in the shortest amount of time fea-sible, where:

- Each robot can only execute one operation per time step;
- There can be no collisions.
- There are no shelves that may be placed on grid cells (which are designated as highways) the project background information

The biggest challenges I had when I began working on this project were many. To overcome these challenges and complete the project, I contacted many additional resources and went to the training sessions in person.

- In the first place, I had difficulty understanding out how the basic instances in-put files were to be used.
- It was unclear to me in the entire section under "Output Description" Because the issue description said that outputs required to be done in the "occur()" way, this project description did not explain why this was the case.
  This is because the project description given for this project used a complicated scoring system, which resulted in inaccurate calculation of how long it takes to complete an order. The reason for this is because as a result, I couldn't find a way to include it into my code.
- Because of the large number of moving components that must be addressed, producing the proper action effects takes a significant amount of time. There were numerous states for each object (robot, shelves, etc.), which resulted in a large number of action effects that needed to be specified.
  By repeatedly reviewing the project description and making certain that I understood it completely each time, I was able to overcome these problems. I also had a look at the "Information for Participants" page on the ASP Challenge 2019 website to see what the original challenge issuers were hoping to find out. Finally, utilizing the notes from Knowledge Representation and Reasoning and viewing the live session re-cordings, I was able to overcome the problems with the assistance of Saba Ahmed's videos.:
- Ascertained that starting values had to be used in order to create the warehousing environment by doing a compari-

son of the five input instance files. I was able to begin working on the blocks world project after watching Saba Ahmed's video, since I now had a better understanding of how to turn input data into data that I could utilize in my ASP.

- Understanding the anticipated output descriptions: The example in Section 4 of the project description, as well as the picture on Page 6 of the project description, were sufficient in providing me with a fundamental understanding of how to construct actions and action effects. • Understanding the anticipated output descriptions:
- Saba Ahmed said in the week 5 video that we may completely ignore this score part since it has nothing to do with this course.

By reviewing previous projects, especially the blocks world assignment, I was able to get a better grasp of the basics of ASP structure and how the logic should be laid out. By using this knowledge, I was able to write out how I wanted the actions (such as robotic shelf-picking and robotic movement) to affect the scenario for each timestamp in the simulation.

The majority of the materials I need to complete this assignment were available to me via a few other sources, and those additional sources were simply essential to assist me in better understanding the challenge goals.

Approach to Solving The Problem

The bulk of my project approach is to clearly and easily lay out each step I must take to get to the desired ultimate state, which is the goal of this project. I used the approach of breaking down the ASP script into its separate components. It worked well for me. The following is a breakdown of the information:

• Processing the input files for the test instance file.

• Making a preliminary layout of the "scenario" by converting the inputs and putting them in it.

Actions were generated by beginning from an initial position and defining a set of possible moves. In addition, I have identified the connections between the timing of different activities.

In order to avoid the robots and shelves from interacting in an illogical manner, I have added state restrictions.

• Action Limitations: I've placed a plethora of restrictions on the activities that may be carried out on my behalf.

If no action(s) is performed, the state(s) would stay unaltered. • Commonsense Rules of Inertia: I developed inertia rules to describe the states that would remain unchanged if no action(s) is taken.

Each of the acts detailed in the previous sections was associated with a specific consequence, which I specified below.

In order to prevent a situation in which there is neither an orderat() nor an orderat(), I defined the goal/end state as when there is neither an orderat() nor an orderat(), despite the fact that this was not required by the problem.

• Output Testing: Although this is not explicitly stated in the script, it is an essential part of the whole project. It was necessary to test all five instances in order to ensure that I was able to get legitimate models rather than poor findings. A manual examination of the outputs was then performed in order to identify any instances of inexplicable behavior.

I was able to concentrate on one section at a time and fully develop each component of the script as a result of adopting this organized, broken-down approach to writing. This approach proved to be the most successful in terms of being comprehensive and requiring the least amount of correction.

Results and Analysis

In this part, I'll go over the results I discovered for the class's five basic sample instances. Before I go into each case, here's a quick rundown of all five cases and the shortest "time" in which I was able to discover at least one satisfiable model.

Simple Instance	Shortest Time (n-1)
Instance1	10
Instance2	11
Instance3	7
Instance4	5
Instance5	7

Table 1 – Stable Models Found Summary

Using my constraints and assertions, as indicated in the table above, each of the five example basic cases may be resolved completely. In any of these smallest time numbers, the starting time zero is not included in any way.

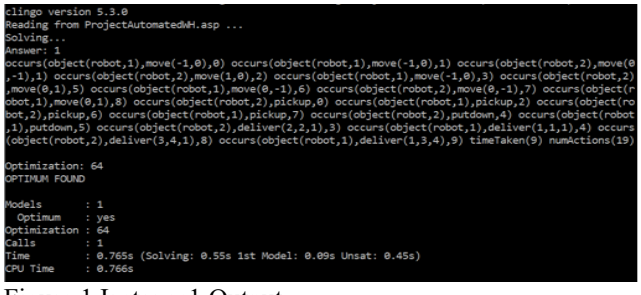

Figure 1 Instance1 Output.

When I was running the Asp script in Clingo, I had to specify both the script and the input instance file at the same time. In addition, I must supply the program with a "n" value, which represents the number of "states" that are allowed to be explored in order to find a solution in this situation. When the number of satisfied models for instance1 (Figure 1) was increased to nine (time taken = eight), there were no satisfied models. When the n number was set to 10, the ASP was able to find a satisfactory model in a reasonable amount of time (time took = 9). Consequently, I was able to complete the first instance of my script in about nine minutes.

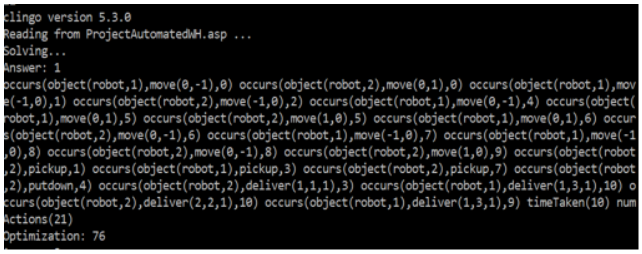

Figure 2-1 Instance 2 Output

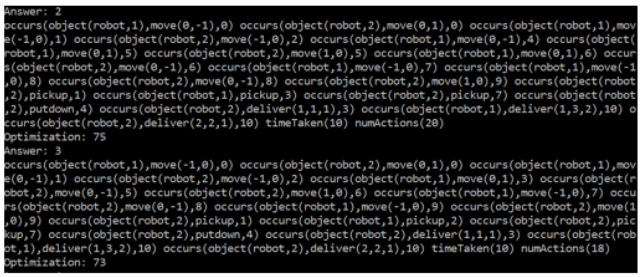

Figure 2-2 – Instance2 Output

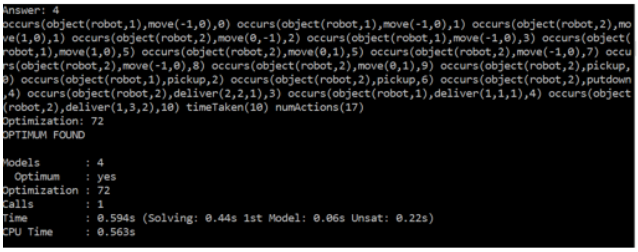

Figure 2-3 – Instance2 Output

With regards to instance 2 (Figure 2) , the ASP was able to construct a satisfied model when the n number was set to 11 (i.e. when the time taken was ten). As a result, I was able to complete the first instance in less than ten minutes using my script.

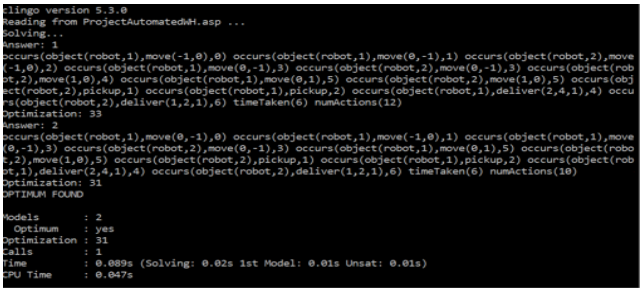

Figure 3 Instance3 Output

When the n number was adjusted to 7, the ASP was able to develop a satisfied model for instance3 (Figure 3) in less than six minutes (time taken = 6). My script was able to complete the first instance in about 6 minutes as a result of

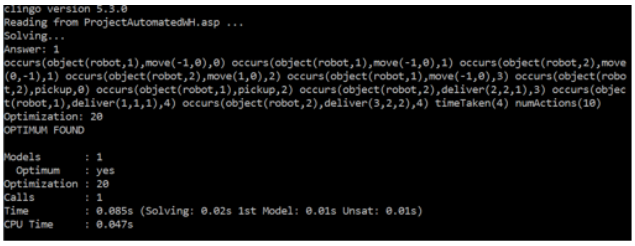

Figure 4 – Instance4 Output

this.

ASP was successful in finding a satisfied model for instance4 (Figure 4) when the number of iterations (n) was set to 5 (time taken = 4). It only took me around four minutes to complete the first instance of my script as a result of this strategy.

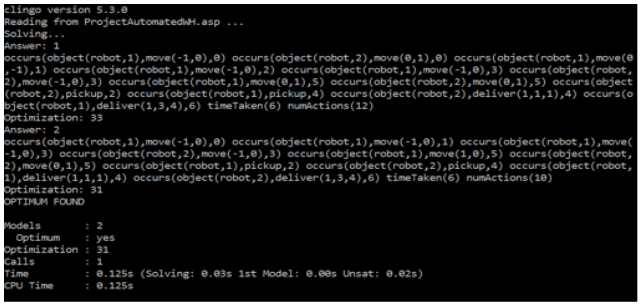

Figure 5 Instance 5 Output

When the n parameter was increased to 7, the ASP was able to rapidly find a satisfied model for instance 5 (Figure 5) (time taken = 6). As a consequence, my script was able to finish the first instance in about six minutes.

Result Analysis:

Because writing the analysis for each situation would take a long time, I'll concentrate on two specific examples. Only instances 1 and 4 will be discussed in this part. The correctness of each occurrence, however, is verified using the same methodology and approach.

An example is as follows: Step 1, which involves getting the results and sorting them by time and robot, requires that the output be delimiting and sorting by time and robot. From the sorted order, we can see that robot 1 performs all of the orders that fall into the "order 1" category, while robot 2 does "orders 2 and 3" as well. After moving from (4,3) to (2,3), robot 1 will arrive to shelf 3, which is located two positions away, in two moves (time = 2). Shelf 3 then moves to the picking station on position (1,3) at a different time and in a different manner. When robot 1 has just

completed delivering a "quantity of 1" for "product 1" to "order 1", the "result" shown is "occurs(object(robot,1),deliver(1,1,1),4". Shelf 3 is the correct step since in the initial state "shelf 3" contains one of "product 1" on it (this is from the inst1.asp file).

Following the same reasoning, robot 2 begins at the position of "shelf 4," which has a "quantity of one" for "product 2." This is required for "order 2" at "picking station 2," and robot 2 will take the quickest route to deliver this order.

Following the remainder of the output, it is obvious that robot 1 delivered "1 of product 1 for order 1 at picking station 1" and "4 of product 3 for order 1 at picking station 1." Robot 2 delivers "1 of product 2 for order 2 at picking station 2" and "1 of product 4 for order 3 at picking station 2" in the meanwhile. All of this was accomplished while adhering to the program's strict guidelines (e.g. only one shelf on a robot, no put-ting down shelves on the highway, etc.). This ASP software was able to optimize this instance and send the necessary items for each order to the proper pickup station in the quickest period possible.

Using the sorted list for instance 4, we can use the same method to verify inst4. Robot 1 successfully delivers “1 of product 1 for order 1 at picking station 1” while robot 2 delivers “1 of product 2 for order 2 at picking station 2” and “2 of product 2 for order 3 at picking station 2”. Robot 2 only needed to pick up “shelf 4” once to satisfy both orders 2 and 3 since they are for the same product and “shelf 4” has enough quantity. This again was achieved without violating any hard constraints in the program.

As can be seen from these two case studies, this ASP is capable of determining the shortest time required to complete the requested orders in the most effective manner.

Conclusion

The goal of this project was to identify the most effective method of completing all of the orders for an automated warehousing facility. A almost comprehensive visualisation of the warehouse is required in order to accomplish this task. When this is supplied by the starting states, it is simple to do by imposing the appropriate hard restrictions on the activities and states that are permitted to occur. It is easy to build up an automated warehouse that can achieve this goal using the many pieces of information supplied originally, such as the goods available and their locations, highway locations, robot beginning places, and so on.
It can be observed from the output of the basic examples that my solution to this issue statement allows the robots to choose the order in which they will complete their tasks by determining which one will allow them to work the most effectively. They are able to accomplish all of this while adhering to the strict state and action limitations. Overall, my evaluation of my ASP is that it is a legitimate and effective solution for the issue statement relating to an automated warehouse.

Opportunities for Future Work

The ideas and information that I have acquired during this project will be useful to me in my future job endeavors. When it comes to grammar, one of the most essential ideas I learnt was the usage of the "implied false if not" statement (double negation). This is so different for me that having to apply it so many times as a restriction in this project has helped me to have a deeper understanding of the concepts involved.
When it comes to tackling difficult real-world problems, Warehouse is a great illustration of how ASP (and KRR) can make a significant difference. Traditional techniques to computer science grow less and less efficient as the complexity of the issue increases. It is only when logic-based functional programming takes the lead that success may be achieved. The whole narrative is built on the concept of limits and how KRR solves complex real-world problems with a simple approach to problem solving. Many applications of ASP (knowledge representation) may be found in the fields of artificial intelligence, Workforce management, intelligent call routing protocols (for contact centers), and decision support centers are some of the fields in which we are now investing our resources., and many other fields of study.
Due to the fact that it supports recursive definitions, aggregates, weight constraints, optimization instructions, and default negations., and external atoms, the ASP has shown to be quite useful in the solution of cross-domain complex issues. With this built-in expressiveness, declarative ASP may be used to address both combinatorial search difficulties (such as planning, diagnostics, and so on) and knowledge-intensive problems (such as knowledge discovery) via declarative programming (such as query answering, explanation generation).In recent years, ASP has been effectively used to a wide range of topics in artificial intelligence and other domains.

References

[1] ASP Challenge of 2019. https://sites.google.com/view/aspcomp2019/

Appendix

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% sort and object declaration
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Count Rows in warehouse Grid
rowCount(RowC):- RowC=#count{Y:init(object(node,I),value(at,pair(X,Y)))}.

% Count Cols in warehouse Grid
colCount(ColC):- ColC=#count{X:init(object(node,I),value(at,pair(X,Y)))}.

% Count Robots
robotCount(RobotC):- RobotC=#count{I:init(object(robot,I),value(at,pair(X,Y)))}.

% Type of node :- Regular , Highway
node(NodeID):- init(object(node,NodeID),value(at,pair(X,Y))).
nodeAt(NodeID,pair(X,Y)):- init(object(node,NodeID),value(at,pair(X,Y))).
highway(NodeID):- init(object(highway,NodeID),value(at,pair(X,Y))).

% Picking Station and where it is at
pickingStation(PickStationID):- init(object(pickingStation,PickStationID),value(at,pair(X,Y))), init(object(node,NodeID),value(at,pair(X,Y))).
pickingStationAt(PickStationID,NodeID):- init(object(pickingStation,PickStationID),value(at,pair(X,Y))), init(object(node,NodeID),value(at,pair(X,Y))).

% Robot
robot(RobotID):- init(object(robot,RobotID),value(at,pair(X,Y))).

% Shelf
shelfDetails(ShelfID):- init(object(shelf,ShelfID),value(at,pair(X,Y))).

% Determine location of Shelf and Robots at time 1
robotLocation(RobotID,object(node,NodeID),1):- init(object(robot,RobotID),value(at,pair(X,Y))), nodeAt(NodeID,pair(X,Y)).
shelfLocation(ShelfID,object(node,NodeID),1):- init(object(shelf,ShelfID),value(at,pair(X,Y))), nodeAt(NodeID,pair(X,Y)).

%Product on Shelf Details
productOnshelfDetails(Product,object(shelf,ShelfID),with(quantity,ProductQuantity),1):- init(object(product,Product),value(on,pair(ShelfID,ProductQuantity))).

% Order details and Product on shelf details at time 1
orderDetails(OrderID,object(node,ND),contains(Product,ProductQuantity),1):- init(object(order,OrderID),value(pickingStation,PKI)), pickingStationAt(PKI,ND), init(object(order,OrderID),value(line,pair(Product,ProductQuantity))).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Actions
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
move(0,1;0,-1;-1,0;1,0).
{robotMove(R,move(DX,DY),T):move(DX,DY)}1:- R=1..RobotC, robotCount(RobotC), T=1..m.
{robotPickUpshelf(R,ShelfID,T):shelfDetails(ShelfID)}1:- R=1..RobotC, robotCount(RobotC), T=1..m.
{robotPutDownshelf(R,ShelfID,T):shelfDetails(ShelfID)}1:- R=1..RobotC, robotCount(RobotC), T=1..m.
```

```
{robotDeliverOrder(R,OrderID,with(ShelfID,PROD,DelQ),T):orderDetails(OrderID,object(node,ND),contains(PROD,OrderQ),T), productOnshelfDetails(PROD,object(shelf,ShelfID),with(quantity,ProductQuantity),T), DelQ=1..ProductQuantity}1:- R=1..RobotC, robotCount(RobotC), T=1..m.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Output
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
occurs(object(robot,R),move(DX,DY),T):-robotMove(R,move(DX,DY),T), T=1..m.
occurs(object(robot,R),pickup,T):-robotPickUpshelf(R,_,T), T=1..m.
occurs(object(robot,R),putdown,T):-robotPutDownshelf(R,_,T), T=1..m.
occurs(object(robot,R),deliver(OrderID,Product,DelQ),T):-robotDeliverOrder(R,OrderID,with(ShelfID,Product,DelQ),T), T=1..m.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Action Constraint
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Two actions cannot occur at the same time
:- occurs(object(robot,R),Action1,T), occurs(object(robot,R),Action2,T), Action1!=Action2 , T=1..m.

%Action Constraint of move
% Robot can move within the defined grid of warehouse
:- robotLocation(R,object(node,ND),T), robotMove(R,move(DX,DY),T), nodeAt(ND,pair(X,Y)), X+DX<1 , T=1..m.
:- robotLocation(R,object(node,ND),T), robotMove(R,move(DX,DY),T), nodeAt(ND,pair(X,Y)), Y+DY<1 , T=1..m.
:- robotLocation(R,object(node,ND),T), robotMove(R,move(DX,DY),T), nodeAt(ND,pair(X,Y)), X+DX>ColC, colCount(ColC) , T=1..m.
:- robotLocation(R,object(node,ND),T), robotMove(R,move(DX,DY),T), nodeAt(ND,pair(X,Y)), Y+DY>RowC, rowCount(RowC) , T=1..m.

% Action Constraint of robotPickUpshelf
% robot can pick up shelf only if it is located on the node containing that shelf
:- robotPickUpshelf(R,S,T), shelfLocation(S,object(node,ND),T), not robotLocation(R,object(node,ND),T), T=1..m.

% Robot cannot pickup a shelf that is held by other robot
:- robotPickUpshelf(R1,S,T), shelfLocation(S,object(robot,R2),T), T=1..m.

%Action Constraint  of robotPutDownshelf
% Robot can put down a shelf only if it has one.
:- robotPutDownshelf(R,S,T), not shelfLocation(S,object(robot,R),T), T=1..m.

% Robot cannot putdown a shelf on a highway
:- robotPutDownshelf(R,S,T), robotLocation(R,object(node,ND),T), highway(ND), T=1..m.

% Robot cannot putdown shelf on picking station
:- robotPutDownshelf(R,S,T), robotLocation(R,object(node,ND),T), pickingStationAt(_,ND), T=1..m.

%Action Constraint  of robotDeliverOrder
% robotDeliverOrder possible only when robot is at picking station
:- robotDeliverOrder(R,OrderID,with(_,PR,_),T), orderDetails(OrderID,object(node,ND),contains(PR,_),T), not robotLocation(R,object(node, ND),T), T=1..m.

% robotDeliverOrder if robot has the correct shelf containing product
:- robotDeliverOrder(R,OrderID,with(ShelfID,PR,_),T), productOnshelfDetails(PR,object(shelf,ShelfID),with(quantity,_),T), not shelfLocation(ShelfID,object(robot,R),T), T=1..m.
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% State Constraint
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Shelf cannot be on a highway.
:- shelfLocation(S,object(node,NodeID), ), highway(NodeID).

% No robot at two nodes
:- not {robotLocation(R,object(node,ND),T):node(ND)}1, robot(R), T=1..m+1.

% Two robots not on the same node
:- not {robotLocation(R,object(node,ND),T):robot(R)}1, node(ND), T=1..m+1.

% Robots cannot swap location
:- robotLocation(R1,object(node,ND1),T), robotLocation(R1,object(node,ND2),T+1), robotLocation(R2,object(node,ND2),T), robotLocation(R2,object(node,ND1),T+1), R1!=R2, T=1..m+1.

% Shelf cannot be on two different robots
%:- not{shelfLocation(S,object(robot,NR),T): robot(NR)}1, shelfDetails(S), T=1..m+1.
:- shelfLocation(S,object(robot,R1),T), shelfLocation(S,object(robot,R2),T), R1!=R2, T=1..m+1.

% Robot cannot have more than one shelf
:- not{shelfLocation(S,object(robot,NR),T): shelfDetails(S)}1, robot(NR), T=1..m+1.

% Shelf cannot be on two nodes
:- not{shelfLocation(S,object(node,ND),T): node(ND)}1, shelfDetails(S), T=1..m+1.

% Two shelf not on the same node
:- not{shelfLocation(S,object(node,ND),T): shelfDetails(S)}1, node(ND), T=1..m+1.

% Robot when carrying a shelf cannot move to a cell which has a shelf already
:- shelfLocation(S1,object(robot,R),T),robotLocation(R,object(node,ND2),T+1), shelfLocation(S2,object(node,ND2),T+1),S1!=S2, not pickingStationAt( ,ND2), T=1..m+1.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Action Effects
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Effect of move action
robotLocation(R,object(node,ND2),T+1):- robotLocation(R,object(node,ND1),T), nodeAt(ND1,pair(X,Y)), nodeAt(ND2, pair(X+DX,Y+DY)), robotMove(R,move(DX,DY),T), T=1..m.

% Effect of robotPickUpshelf
shelfLocation(S,object(robot,R),T+1):- robotPickUpshelf(R,S,T), shelfLocation(S,object(node,ND),T), robotLocation(R,object(node,ND),T), T=1..m.

% Effect of robotPutDownshelf
shelfLocation(S,object(node,ND),T+1):- robotPutDownshelf(R,S,T), shelfLocation(S,object(robot,R),T), robotLocation(R,object(node,ND),T), T=1..m.

% Effect of robotDeliverOrder
orderDetails(OrderID,object(node,ND),contains(PROD,OrdQ-DelQ),T+1):- robotDeliverOrder(R,OrderID,with(ShelfID,PROD,DelQ),T), orderDetails(OrderID,object(node,ND),contains(PROD,OrdQ),T), T=1..m.
```

```
productOnshelfDetails(PROD,object(shelf,ShelfID),with(quantity,ProductQuantity-DelQ),T+1):- robotDeliverOrder(R,OrderID,with(ShelfID,PROD,DelQ),T), productOnshelfDetails(PROD,object(shelf,ShelfID),with(quantity,ProductQuantity),T), T=1..m.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Common Law of Intertia
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
robotLocation(R,object(node,ND),T+1):- robotLocation(R,object(node,ND),T), not robotMove(R,move( , ),T), T=1..m.
shelfLocation(S,object(node,ND),T+1):-shelfLocation(S,object(node,ND),T), not robotPickUpshelf( ,S,T), T=1..m.
shelfLocation(S,object(robot,R),T+1):-shelfLocation(S,object(robot,R),T), not robotPutDownshelf(R,S,T), T=1..m.
orderDetails(OrderID,object(node,ND),contains(PROD,OrdQ),T+1):- orderDetails(OrderID,object(node,ND),contains(PROD,OrdQ),T), productOnshelfDetails(PROD,object(shelf,ShelfID),with(quantity,ProductQuantity),T), not robotDeliverOrder( ,OrderID,with(ShelfID,PROD, ),T), T=1..m.
productOnshelfDetails(PROD,object(shelf,ShelfID),with(quantity,ProductQuantity),T+1):- productOnshelfDetails(PROD,object(shelf,ShelfID),with(quantity,ProductQuantity),T), not robotDeliverOrder( , ,with(ShelfID,PROD, ),T), T=1..m.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Goal state
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
:- not orderDetails(OrderID,object(node, ),contains(PR,0),m+1), orderDetails(OrderID,object(node, ),contains(PR, ),1).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Calculate Time and action
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
actions(N) :- N=#count{1,R,A,T:occurs(R,A,T)}.
time(N) :- N=#count{T:occurs(R,A,T)}.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Optimization
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Minimize time taken to fulfill order
%#minimize{T:occurs(R,A,T)}.
#minimize { 1,R,A,T : occurs(R,A,T)}.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Show Output
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
#show occurs/3.
#show actions/1.
#show time/1.
```