Individual Project Report

Abhishek Patil
Arizona State University
aspati12@asu.edu

Abstract

This document highlights my automated warehouse project. It starts with explaining the problem and then gives some background on the tools required. Then, it discusses the approach to solution with some issues. Later, it shows results to given test cases. Finally, it explains opportunities of future work and gives the entire code in appendix.

Problem Statement

The Automated Warehouse Scenario, as the name suggests, involves a warehouse where robots need to pick up shelves with specific products and deliver them to specified picking station. This scenario is becoming increasingly popular as it is efficient at processing orders and maintaining a better workplace in terms of health and safety (Lowe 2020).

The problem defines the warehouse as a rectangular area with grids. Initially, robots, shelves, and picking stations are allocated to specific grids. The robots must navigate around the warehouse, completing all orders by delivering specified number of products (using the shelves) to a given picking station.

Due to the reality of the situation, there are physical constraints. As examples, two robots cannot collide, two shelves cannot be on same grid, and so on. Something to note however, is that the robots are considered flat and thus can move under shelves without carrying a shelf.

The goal of this project is to ensure that the robots complete all orders without breaking constraints by using minimum time possible.

Project Background

The Automated warehouse scenario is an NP-hard problem (similar to Travelling Salesman problem) as mentioned in the thesis work by van Rensburg (2019). As mentioned in his paper, Vladimir Lifschitz (2008) explains Answer Set Programming (ASP) as a declarative programming made for NP-hard problems. Declarative programming focuses on describing the desired result without telling how to achieve it (Travis 2020). Thus, Answer Set Programming with Clingo provides a single system to write and execute code to find possible solutions.

In addition to using a convenient programming language, ASP has various constructs and some well-defined ways to incorporate actions in a program. The idea is to write rules that define an initial state, perform actions to change state and come up with a solution based on a goal.

Project Approach

The approach described here goes through the application I created. The entire code is presented in Appendix. The explanation tries to be precise and beginner friendly.

I started this problem by understanding the initializations first. There is a square area divided into nodes (grids) as in figure 1. The top-left node labelled 1 in figure 1 starts at x=1, y=1, node 2 is at x=2, y=1 and so on. In terms of the ASP rule, depicting node 1 for example is as follows: *init(object(node,1),value(at,pair(1,1)))*. Here 'init' stands for initialization, 'object(node,1)' identifies the node 1 in the figure and 'value(at,pair(1,1))' gives the coordinates of x=1, y=1 to node 1. Nodes 2 through 16 are defined in a similar way.

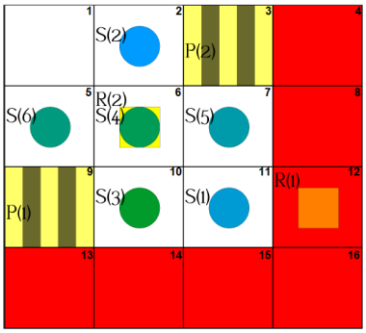

Figure 1. An example of initial state of 4x4 square warehouse (Gebser, Obermeier 2019).

Some nodes in the warehouse could also be marked as highways where no shelf can be put down (marked as red in figure1). In ASP the rule for highway at node 4 is:
init(object(highway,4),value(at,pair(4,1))).

There are also picking stations (P(1) and P(2) in figure 1) where robots will approach these stations with respective shelf to perform the deliver action. Additionally, there are objects that change over time through different actions: robots, shelves, products, and orders. These objects are called fluents. It makes sense to represent initial states for fluents as values at time step T=0. I set up robots, shelves, their locations, items (products) with their quantities and orders with item quantities at time T=0.

Moving on to the main approach, it was to stick to the examples of programming actions in ASP, namely blocks world and monkey bananas problem. The idea is to create 4 major sections: state descriptions, effects and preconditions of actions, domain independent axioms and setting the goal.

Firstly, for the state descriptions, I have defined a new state value called robotHasShelf as follows:
robotHasShelf(RID, SID, B, T). This shows if the robot RID does/does not have a particular shelf SID based on boolean B at time T. Using this, I have defined all robots to not have a shelf at the beginning, so every RID and SID combination is set to B='f' at T=0.

Secondly, for the effects and preconditions of actions there are the following:

- The 'move' action is specified as follows:

occurs(object(robot,RID),move(DX,DY),T) where robot RID moves some DX in x-axis and DY in y-axis at time T with respect to its node location at T-1. (DX,DY) belongs to {(0,-1), (1,0), (0,1), (-1,0)} which represent robot movement in North, East, West and South directions.
Preconditions: robot cannot go to a node that is not defined in initializations. Additionally, no more than 2 robots or shelves are on any node at same time. Also, robots not swapping their locations is done by checking if these robots interchange node locations for consecutive times.

- The 'pickup' action is specified as follows:

occurs(object(robot,RID),pickup,T) where the robot pickups a shelf present at its node at time T-1. This changes *robotHasShelf(RID, SID, f, T)* to *robotHasShelf(RID, SID, t, T)* for the shelf SID at RID's location.
Preconditions: robot cannot pickup if the number of shelves at its node is not equal to 1. Also, it cannot pickup if it already has a shelf i.e. *robotHasShelf(RID, SID, t, T).*

- The 'putdown' action is specified as follows:

occurs(object(robot,RID),putdown,T) where the robot puts down the shelf it was carrying at T-1. *robotHasShelf(RID, SID, t, T)* changes to *robotHasShelf(RID, SID, f, T)* for the shelf SID carried at T-1.
Preconditions: robot cannot putdown if the count of shelves it has is not equal to 1. Also, the robot cannot putdown a shelf if it is on a highway node at T-1.

- The 'deliver' action is specified as follows:

occurs(object(robot,RID),deliver(OID,IID,U),T) where the robot RID delivers U number of item IID in the order OID at time T by carrying a shelf with item IID at T-1. I introduced two values *order_attr(OID, IID, OU, T)* and *item_attr(IID, SID, SU, T)* where OU and SU are quantities of item IID in orders and shelves respectively. These values change to *order_attr(OID, IID, OU-U, T)* and *item_attr(IID, SID, SU-U, T)* when this action occurs. One of my two major issues was long execution times and was solved by ensuring that the values SU-U and OU-U did not go below 0.
Preconditions: robot cannot deliver if it is not at the picking station described in the order. It also does not deliver items if it does not carry the shelf with respective item or has one but with 0 quantities. Also, avoid delivering items that have already been delivered in the order i.e. IIDs where *order_attr(OID, IID, 0, T).*

Along with the above 4 actions, we need to disallow concurrent actions. We disallow move and pickup, move and putdown, move and deliver, and putdown and deliver. Also, a robot cannot deliver different orders or different items at same time T. Not having this constraint caused the other issue where the robot delivered different items in same time T.

Thirdly, under the domain independent axioms, I defined the following:

- Exogenous fluents: This means that we define all possible initial locations of the robots and shelves. By being exogenous, Clingo outputs solutions for every possible robot and shelf locations combinations. In ASP, the rule for exogenous robot locations is:

1{loc(robot, RID, NID, 0) : node(NID,_,_)}1 :- robot(RID).
Here, it can be read as, 'for every robot RID choose exactly 1 out of a set of all possible nodes NID at time T=0'.

- Uniqueness and Existence of fluents: This is an important part of actions in ASP. The value of fluent at any time T in the program should always be unique and exist. If such a rule is 'not' followed, then Clingo discards the solution. An example of this is for the robot location:

:- not 1{loc(robot, RID, NID, T) : node(NID,_,_)}1, robot(RID), T=1..m. It can be read as, 'for every robot RID at a time T if there does 'not' exist exactly 1 node NID as the location then just discard it. In a similar way, it applies to other fluents.

- Exogenous actions: This means that each robot performs every kind of action at first. Clingo creates different scenarios for each kind of action. However, with the preconditions defined, only sensible scenarios of those actions are maintained. Out of all the actions, 'deliver' action is most complicated. It requires a specific order line given by *or-*

der_attr(OID, IID, OU, T) and a specific shelf with the respective item given by *item_attr(IID, SID, SU, T)*. It naturally makes sense to use the lesser of the two item quantities between OU and SU and thus it creates the action: *{occurs(object(robot,RID),deliver(OID,IID,U),T)}* where U is min(OU, SU). The braces make Clingo create a scenario where this action happens and another where it does not. This is applied for every order given in initialization.

In the case of the move action, each robot has 5 scenarios: do not move, move North, East, West and South. For pickup and putdown actions it is as simple as happen or do not happen.

• Common sense law of inertia: This means that if an action did not occur then keep the positions the same from previous time T-1. For example, the robot location stays if it does not move and the ASP rule is: *{loc(robot, RID, NID, T)} :- loc(robot, RID, NID, T-1), T = 1..m.* Here for every robot RID it gets the location at time T-1 and sets that value for next time step. The braces mean Clingo creates another scenario where the default location is not kept. In such scenario, if an action like move takes place, it forgets about the default location because a robot cannot have multiple locations at same time (from the uniqueness and existence rule mentioned before). Similarly, other fluents keep default values without the action.

The final part of code sets the goal. It is a simple check to see that for all orders OID and items IID the order quantity OU has reached 0 in *order_attr(OID, IID, OU, T)*. If not then Clingo sees the following rule as violation and discards such scenarios: *:- not 1{order_attr(OID, IID, 0, m)}1, order(OID, _), order_attr(OID, IID, _, 0).* Here the use of braces ensures we define this rule for every order OID and respective IID.

After the goal was successfully reached, I optimized the total time steps taken by using the following in Clingo: *#minimize{T,X,Y:occurs(X,Y,T)}*. Here it takes minimum of the sum of total time step values starting from T=0 to T=m but only for actions that occurred in each scenario.

Main Results and Analysis

To show the test cases performed, I have formatted them into tables which show different positions and quantities of objects. The output is screenshot exactly as seen in clingo.

The general command line to run these test cases is: *clingo <init_file>.lp aws.lp -c m=<timestep> 0* where aws.lp is the main application, <init_file> is the input initialization file name and <timestep> is the minimum timesteps needed to come up with a feasible solution.

The value (A,B) in the table gives location (x,y) (as seen in figure 1) for picking stations, robots and shelves. The same format is used to show that item is on shelf A with quantity B. For orders, it shows the picking station, item and quantity needed tuple. The node and highway initializations have been omitted because they are exactly same for all the test cases as seen in figure 1.

• Test case 1: This is the most complex test case with multiple orders and items as seen in Table 1.

Object	Value
Picking Stations (x2)	PS1-(1,3), PS2-(3,1)
Robots (x2)	R1-(4,3), R2-(2,2)
Shelves (x6)	S1-(3,3), S2-(2,1), S3-(2,3), S4-(2,2), S5-(3,2), S6-(1,2)
Items (x4)	I1-(S3,1), I2-(S4,1), I3-(S6,4), I4-(S5,1) & (S6,1)
Orders (x3)	O1-(PS1,I1,1) & (PS1,I3,4), O2-(PS2,I2,1),O3-(PS2,I4,1)

Table 1. Test case 1.

Output: 13 timesteps for output (figure 2). At the 11th timestep, robot 1 moves to highway node but does not put down. This way it makes space for robot 2 to deliver.

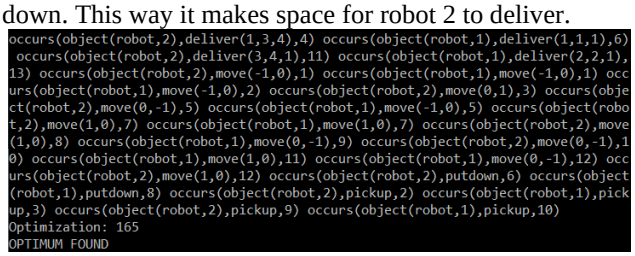

Figure 2. Output for test case 1 (m=13).

• Test case 2: This test case has more items and each of them occur in some order as seen in table 2.

Object	Value
Pick Stations (x2)	PS1-(1,3), PS2-(3,1)
Robots (x2)	R1-(4,3), R2-(2,2)
Shelves (x5)	S1-(3,3), S2-(2,1), S3-(2,3), S4-(2,2), S5-(3,2)
Items (x3)	I1-(S3,1),I2-(S4,1),I3-(S5,3)
Orders (x2)	O1-(PS1,I1,1) & (PS1,I3,2), O2-(PS2,I2,1)

Table 2. Test case 2.

Output: 11 timesteps for output (figure 3). Robot 1 and 2 work on O1 based on where the item is located. Robot 2 additionally takes care of order O2.

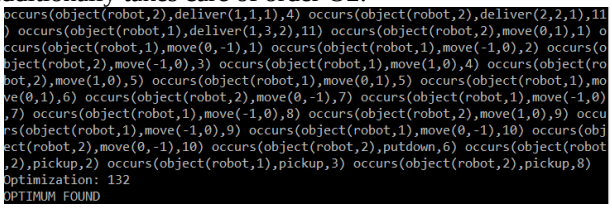

Figure 3. Output for test case 2 (m=11).

• Test case 3: This test case has multiple orders to just one picking station PS1 as seen in table 3.

Object	Value
Pick Stations (x1)	PS1-(3,1)
Robots (x2)	R1-(4,3), R2-(2,2)
Shelves (x6)	S1-(3,3), S2-(2,1), S3-(2,3), S4-(2,2), S5-(3,2), S6-(1,2)
Items (x4)	I1-(S3,1), I2-(S4,1), I3-(S6,4), I4-(S5,1) & (S6,1)
Orders (x3)	O1-(PS1,I2,1), O2-(PS1,I4,1)

Table 3. Test case 3.

Output: 7 timesteps needed (figure 4). Robot 1 handles O2 and makes space for robot 2 to deliver O1 at same PS1.

```
occurs(object(robot,1),deliver(2,4,1),5) occurs(object(robot,2),deliver(1,2,1),7)
 occurs(object(robot,1),move(0,-1),1) occurs(object(robot,1),move(-1,0),2) occurs
(object(robot,1),move(0,-1),4) occurs(object(robot,2),move(1,0),4) occurs(object(
robot,2),move(0,-1),6) occurs(object(robot,1),move(1,0),6) occurs(object(robot,2)
,pickup,1) occurs(object(robot,1),pickup,3)
Optimization: 39
OPTIMUM FOUND
```

Figure 4. Output for test case 3 (m=7).

• Test case 4: This test case has same items of two different orders at same picking stations (table 4).

Object	Value
Pick Stations (x2)	PS1-(1,3), PS2-(3,1)
Robots (x2)	R1-(4,3), R2-(2,2)
Shelves (x6)	S1-(3,3), S2-(2,1), S3-(2,3), S4-(2,2), S5-(3,2), S6-(1,2)
Items (x2)	I1-(S3,1), I2-(S4,3)
Orders (x3)	O1-(PS1,I1,1), O2-(PS2,I2,1), O3-(PS2,I2,2)

Table 4. Test case 4.

Output: 10 timesteps (figure 5). Robot 1 delivers different items at same PS at two consecutive times T=7 and T=8.

```
occurs(object(robot,1),deliver(3,2,2),7) occurs(object(robot,1),deliver(2,2,1),8)
 occurs(object(robot,2),deliver(1,1,1),10) occurs(object(robot,2),move(0,-1),1) o
ccurs(object(robot,1),move(-1,0),1) occurs(object(robot,1),move(-1,0),2) occurs(o
bject(robot,2),move(-1,0),3) occurs(object(robot,1),move(0,-1),3) occurs(object(r
obot,2),move(0,1),5) occurs(object(robot,1),move(0,-1),5) occurs(object(robot,2),
move(0,1),6) occurs(object(robot,1),move(1,0),6) occurs(object(robot,2),move(1,0)
,7) occurs(object(robot,2),move(-1,0),9) occurs(object(robot,2),putdown,4) occurs
(object(robot,2),pickup,2) occurs(object(robot,1),pickup,4) occurs(object(robot,2
),pickup,8)
Optimization: 91
OPTIMUM FOUND
```

Figure 5. Output for test case 4 (m=10).

• Test case 5: Test case with only one order (table 5).

Object	Value
Pick Stations ‘PS’(x1)	PS1-(1,3)
Robots ‘R’(x2)	R1-(4,3), R2-(2,2)
Shelves ‘S’(x6)	S1-(3,3), S2-(2,1), S3-(2,3), S4-(2,2), S5-(3,2), S6-(1,2)
Items ‘I’(x4)	I1-(S3,1), I2-(S4,1), I3-(S6,4), I4-(S5,1) and (S6,1)
Orders ‘O’(x3)	O1-(PS1,I1,1) & (PS1,I3,4)

Table 5. Test case 5.

Output: 6 timesteps (figure 11). Robot 1 waits at T=4 for robot 2 to finish another delivery at same picking station.

```
occurs(object(robot,2),deliver(1,3,4),4) occurs(object(robot,1),deliver(1,
1,1),6) occurs(object(robot,2),move(-1,0),1) occurs(object(robot,1),move(-
1,0),1) occurs(object(robot,1),move(-1,0),2) occurs(object(robot,2),move(0
,1),3) occurs(object(robot,2),move(0,1),5) occurs(object(robot,1),move(-1,
0),5) occurs(object(robot,2),pickup,2) occurs(object(robot,1),pickup,3)
Optimization: 32
OPTIMUM FOUND
```

Figure 11. Output for test case 5 (m=6).

For the given test cases, I checked correctness by manually going through solutions, ensuring all rules were followed as expected and minimum timestep was achieved.

Conclusion

In conclusion, the amount of planning and following the approach resembling that of other examples of ASP actions has helped me create a sound application that has the proper rules to simulate the warehouse scenario and give possible optimal solutions. It is satisfying to have been able to work on this and come up with a complete solution as far as the test cases are concerned.

Opportunities for Future Work

The automated warehouse scenario here is just a simple case and a good steppingstone to solving real world problems. There are many companies worldwide that have warehouses and it can be very difficult to keep track of inventory, do redundant activities, and perform efficiently without automation. Thus, it can be a great opportunity to solve specific company scenarios using ASP with Clingo.

References

Gebser M.; Obermeier P. 2019. ASP Challenge Problem: Automated Warehouse Scenario. *ASP Challenge 2019 Problem Domains.*

Lifschitz, V. 2008. What is Answer Set Programming? *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence (2008)*.

Lowe, H. 2020. Warehouse Automation: Leveraging Automated Warehouse Systems. *SelectHub*.

Travis, K. 2020. The Complete guide to Declarative programming. *CapitalOne.*

van Rensburg, L. J. Artificial Intelligence for Warehouse Picking Optimization – An NP-Hard Problem. *Uppsala University.*

Appendix

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% symbols used in the code
% identity of any object      -ID
% node ID                     -NID
% highway ID                  -NID
% pickingStation ID           -PSID
% robot ID                    -RID
% shelf ID                    -SID
% order ID                    -OID
% item/product ID             -IID
% item quantity               -U
% time step                   -T (goes till m: max time step)
% row value of grid           -X
% column value of grid        -Y
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%
% sort and object declaration
% (convert given initializations
% into a favorable representation)
%%%%%%%%%%%%%%%%%
boolean(t;f).
% define all nodes and highways
node(NID, X, Y) :- init(object(node,NID),value(at,pair(X,Y))).
highway(NID, X, Y) :- init(object(highway,NID),value(at,pair(X,Y))).

% define all picking stations with respective node ID
pickingStation(PSID, NID) :- init(object(pickingStation,PSID),value(at,pair(X,Y))), node(NID, X, Y).

% define all robots ...
robot(RID) :- init(object(robot,RID),value(at,pair(X,Y))).
% ... and their locations at T=0
loc(robot, RID, NID, 0) :- init(object(robot,RID),value(at,pair(X,Y))), node(NID, X, Y).

% define all shelves ...
shelf(SID) :- init(object(shelf,SID),value(at,pair(X,Y))).
% ... and their locations at T=0
loc(shelf, SID, NID, 0) :- init(object(shelf,SID),value(at,pair(X,Y))), node(NID, X, Y).

% define all products (calling it item I) ...
item(IID) :- init(object(product,IID),value(on,pair(SID,U))).
% ... and their quantity (U) and locations (SID) at T=0 - call these item attributes
item_attr(IID, SID, U, 0) :- init(object(product,IID),value(on,pair(SID,U))).

% define all orders and their picking stations ... (exactly one picking station per order)
order(OID, PSID) :- init(object(order,OID),value(pickingStation,PSID)).
% ... and their lines depicting unique item and quantity combo at T=0 - call these order attributes
```

```
order_attr(OID, IID, U, 0) :- init(object(order,OID),value(line,pair(IID,U))).

%%%%%%%%%%
% state description
%%%%%%%%%%
% initially, no robot carries any shelf
1{robotHasShelf(RID, SID, f, 0)}1 :- robot(RID), shelf(SID).
% if a robot is carrying shelf, shelf location changes at that specific time
loc(shelf, SID, NID, T) :- robotHasShelf(RID, SID, t, T), loc(robot, RID, NID, T), T=0..m.

%%%%%%%%%%%%%%%%%%%%
% effect and preconditions of action
%%%%%%%%%%%%%%%%%%%%
% effect of robot move
loc(robot, RID, TO_NID, T) :- occurs(object(robot,RID),move(DX,DY),T), loc(robot, RID, NID, T-1), node(NID, X, Y), node(TO_NID, X+DX, Y+DY).
% if a node is not present to move onto then it is violation
:- occurs(object(robot,RID),move(DX,DY),T), loc(robot, RID, NID, T-1), node(NID, X, Y), not node(_, X+DX, Y+DY).

% no more than one robot on one node at any time
:- 2{loc(robot, RID, NID, T) : robot(RID)}, node(NID,_,_), T = 0..m.
% cannot swap positions with another robot
:- loc(robot, R1ID, N1ID, T), loc(robot, R2ID, N2ID, T), loc(robot, R1ID, N2ID, T-1), loc(robot, R2ID, N1ID, T-1), R1ID!=R2ID, N1ID!=N2ID, T = 1..m.
% cannot move to a node having a shelf - ensure no two shelves at same node
% (if robotHasShelf then it is ensured that shelf location gets updated every time T)
% (this also ensures that a robot with shelf can't reach another shelf to pickup )
:- loc(shelf, S1ID, NID, T), loc(shelf, S2ID, NID, T), S1ID!=S2ID, T=0..m.

% effect of robot pickup
robotHasShelf(RID, SID, t, T) :- occurs(object(robot,RID),pickup,T), loc(robot, RID, NID, T-1), loc(shelf, SID, NID, T-1).
% cannot pickup if robot already has shelf
:- occurs(object(robot,RID),pickup,T), robotHasShelf(RID, _, t, T-1).
% if robot picks up at a location where no shelf is present
:- occurs(object(robot,RID),pickup,T), loc(robot, RID, NID, T-1), #count{SID : loc(shelf, SID, NID, T-1)}!=1.

% effect of robot putdown
robotHasShelf(RID, SID, f, T) :- occurs(object(robot,RID),putdown,T), loc(robot, RID, NID, T-1), loc(shelf, SID, NID, T-1).
% cannot putdown if robot does not already have a shelf
:- occurs(object(robot,RID),putdown,T), #count{SID : robotHasShelf(RID, SID, t, T-1)}!=1.
% cannot putdown at a node which is also a highway
:- occurs(object(robot,RID),putdown,T), loc(robot, RID, NID, T-1), highway(NID, X, Y).

% effect of robot deliver
% order item update
order_attr(OID, IID, OU-U, T) :- occurs(object(robot,RID),deliver(OID,IID,U),T), order_attr(OID, IID, OU, T-1), OU-U>=0.
% shelf item update
item_attr(IID, SID, SU-U, T) :- occurs(object(robot,RID),deliver(OID,IID,U),T), item_attr(IID, SID, SU, T-1), SU-U>=0.
% cannot deliver if at the wrong picking station for the order
```

```
:- occurs(object(robot,RID),deliver(OID,IID,U),T), order(OID, PSID), pickingStation(PSID, N1ID), loc(robot, RID, N2ID, T-1), N1ID!=N2ID.
% cannot deliver if order with an item has 0 quantity needed
:- occurs(object(robot,RID),deliver(OID,IID,U),T), order_attr(OID, IID, 0, T-1).
% cannot deliver if robot is not carrying the shelf for that item
:- occurs(object(robot,RID),deliver(OID,IID,U),T), #count{SID : item_attr(IID, SID, _, T-1), robotHasShelf(RID, SID, t, T-1)}=0.
% cannot deliver from a shelf with item at 0 quantity
:- occurs(object(robot,RID),deliver(OID,IID,U),T), item_attr(IID, SID, 0, T-1).

% disallow concurrent actions
:- occurs(object(robot,RID),move(DX,DY),T), occurs(object(robot,RID),pickup,T).
:- occurs(object(robot,RID),move(DX,DY),T), occurs(object(robot,RID),putdown,T).
:- occurs(object(robot,RID),move(DX,DY),T), occurs(object(robot,RID),deliver(OID,IID,U),T).
:- occurs(object(robot,RID),putdown,T), occurs(object(robot,RID),deliver(OID,IID,U),T).
% disallow delivering multiple items at once
:- occurs(object(robot,RID),deliver(O1ID,I1ID,U1),T), occurs(object(robot,RID),deliver(O2ID,I2ID,U2),T), I1ID!=I2ID.
% also disallow delivering different orders at same time
:- occurs(object(robot,RID),deliver(O1ID,I1ID,U1),T), occurs(object(robot,RID),deliver(O2ID,I2ID,U2),T), O1ID!=O2ID.

%%%%%%%%%%%%%%%%
% domain independent axioms
%%%%%%%%%%%%%%%%
% fluents are initially exogenous
% all possible robot locations (i.e. nodes)
1{loc(robot, RID, NID, 0) : node(NID,_,_)}1 :- robot(RID).
% all possible shelf locations (i.e. nodes)
1{loc(shelf, SID, NID, 0) : node(NID,_,_)}1 :- shelf(SID).

% uniqueness and existence of value constraints
% each robot has exactly one location at any time
:- not 1{loc(robot, RID, NID, T) : node(NID,_,_)}1, robot(RID), T=1..m.
% each shelf has exactly one location at any time
:- not 1{loc(shelf, SID, NID, T) : node(NID,_,_)}1, shelf(SID), T=1..m.
% each robot cannot 'have' and 'not have' a shelf a time T
:- not 1{robotHasShelf(RID, SID, B, T) : boolean(B)}1, robot(RID), shelf(SID), T=1..m.
% each order item has only one value for the quantity possible at a given time T
:- not 1{order_attr(OID, IID, OU, T)}1, order(OID,_), order_attr(OID,IID,_,0), T=1..m.
% each shelf item has only one value for the quantity possible at a given time T
:- not 1{item_attr(IID, SID, SU, T)}1, item(IID), item_attr(IID,SID,_,0), T=1..m.

% actions are exogenous - only start at T=1
% move action - perform no moves or 1 of the 4 moves for each robot
{occurs(object(robot,RID),move(DX,DY),T) : DX=-1..1, DY=-1..1, |DX|!=|DY|}1 :- robot(RID), T = 1..m.
% pickup action - perform pickup action for all robots or don't perform
{occurs(object(robot,RID),pickup,T)} :- robot(RID), T = 1..m.
% putdown action - perform putdown action for all robots or don't perform
{occurs(object(robot,RID),putdown,T)} :- robot(RID), T = 1..m.
% deliver action - perform deliver action for all robots or don't perform at all
% OU - quantity in order, SU - quantity on shelf, we deliver the minimum of OU and SU
```

```
{occurs(object(robot,RID),deliver(OID,IID,OU),T)} :- robot(RID), order(OID,_), order_attr(OID, IID, OU, T-1), item_attr(IID, _, SU, T-1), OU<=SU, T = 1..m.
{occurs(object(robot,RID),deliver(OID,IID,SU),T)} :- robot(RID), order(OID,_), order_attr(OID, IID, OU, T-1), item_attr(IID, _, SU, T-1), SU<OU, T = 1..m.

% commonsense law of inertia
% robot staying by default
{loc(robot, RID, NID, T)} :- loc(robot, RID, NID, T-1), T = 1..m.
% shelf staying by default
{loc(shelf, SID, NID, T)} :- loc(shelf, SID, NID, T-1), T = 1..m.
% robot having shelf is true or false in the next time step by default - B for boolean
{robotHasShelf(RID, SID, B, T)} :- robotHasShelf(RID, SID, B, T-1), T = 1..m.
% order item quantity is carried on by default
{order_attr(OID, IID, OU, T)} :- order_attr(OID, IID, OU, T-1), T = 1..m.
% shelf item quantity is carried on by default
{item_attr(IID, SID, SU, T)} :- item_attr(IID, SID, SU, T-1), T = 1..m.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% GOAL :- all original orders with their items have reached 0 quantity
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
:- not 1{order_attr(OID, IID, 0, m)}1, order(OID, _), order_attr(OID, IID, _, 0).
% minimize the time taken - minimize on the sum of T in different stable models
#minimize{T,X,Y:occurs(X,Y,T)}.

#show occurs/3.
```