CSE 579 Project Milestone 4 – Individual Project Report
Automated Warehouse Scenario

Sang-Hun Sim
School of Computing, Informatics, and Decision Systems Engineering
ssim6@asu.edu

Abstract

In the individual project report, I suggested the solution to the problem of automated warehouse. It mainly deals with transition system that is affected by robot's action and the methodology to reach to the desired status. All the definitions and hard constraints for Automated Warehouse are illustrated in terms of Knowledge Representation and Reasoning (KRR). The purpose of this scenario is to find the minimized time for robots to fulfill the order. Given input data, all objects must be defined, and it is required for some rules and conditions to be added to establish the scenario which is as close to the reality as possible. Each step to approach to the solutions are explained as well as the analysis of results is described. I ended up finding satisfiable results for all the five inputs.

Problem Statement

In the automated warehouse scenario, robots deliver products and bring them into picking stations to fulfill orders by horizontally and vertically moving through grid-shaped cells. Robots transfer the shelves containing the products to match them to picking stations. Robots can move through path below the shelves and pick them up since their body shape is flat. The goal of this scenario is to deliver all the products specified in order state to the picking stations.

At the beginning, I got five instances which set initial status of elements. These are considered as our input. Each element has different format inside of 'init' function. For the output, we first need to consider the robot's actions. There are four of them defined, moving, picking up the shelf, putting it down, and delivering. Robot's actions are the required output of this project and must be expressed in 'occur' function.

There are also needed rules(constraints) in terms of robot's actions and status. These constraints make this scenario be closer to the reality and be more practical in working station. In the description document, for example, a robot cannot pass under the shelves when it handles another shelf already. So, it firstly ought to be moved out of the way to put down the handled shelf. Also, the time is counted in step where robots exploit exactly one action (or do nothing). In other words, robot take a turn to generate one action. Also, there must not be the collision between robots, while they move around, pick up, and put down the shelf, deliver products, or remain idle. Their moves performed via one step from another step to the next. Lastly, the highways can be assigned on grid cells so that no shelves put down on them. These conditions mentioned so far are written in the description document. I have found more constraints during programming and added them into file.

Background

I adopted the Answer Set Program (ASP) as a key tool for solving problems in this scenario. I learned all about the ASP need for this project from the course CSE 579. A Reason for choosing the topic of Automated Warehouse is that I got inspiration on the idea of moving blocks with robots from the Module 5 in this course.

In Module 5, it introduces how to reason about actions. I think this project is also based on transition system in that the goal state is achieved by robot's actions. It also introduces the concept of Fluent. In this project, fluent can be whether a shelf is picked up by a robot or the location of the robot. And every fluent tends to maintain same status by default.

Moreover, it tells us the way of designing a transition system. Atoms are used to express each element the effect of executing actions will be described with it. I could figure out how to define each element with atom and express actions and its effect in timely manner.

I found that ASP is so powerful language that I can easily set hard constraints, with which program must obey. In a

brief sense, I can set the overall environment and add required constraints so that all the elements will follow the rules. Regarding creation of such constraints, I had to consider the complexity of algorithm that I devised, logical contradictory among constraints, and the degree of optimization suggested.

To get overall concept of the project, I used the lecture notes from CSE 579. For deeper understanding of programming, I referred to Vladimir's book, *Answer Set Programming*, which allowed me to collect ideas of ASP programming and its principles. I also referred to *Artificial Intelligence, A Modern Approach* to get a sense of reasoning procedures.

Approaches to Solve

Sketching the Program

Before I embarked on writing codes, I investigated instance files. I checked the type of objects and fluents and decided necessitated parameters of each of them. Then, I envisioned what the automated warehouse would look like. As per instance file, the warehouse is on 4×4 grid and each fluent resides on their coordinate. Then I figured out how to set fluents' initial status from given instances. And since there are four types of occur function, I determined that there are four actions that robot can commit. I also made the list of required constraint so that I can include them in my code easily.

Fluents and Objects

The next thing to do is to deal with the domain-independent axioms. I found that fluents are not initially exogenous since they must follow the status of initial instances. Thus, I had to convert input into the format for parsing and declaring them as instances and fluents. Then I defined the atoms depicting its status and an object's identification with the key value and supportive parameter such as coordination or corresponding objects. For instance, the atom for the object 'node' has 'Nid' as unique value for each node and 'pair(X,Y)', the coordination.

Most of other objects are composed of their key value and the information of the node where they currently step on. I also constructed the counting functions for objects to be used for defining constraints and other rules. For example, information about the number of entire nodes was needed to specify the goal states.

Transitions

I specified transition statements to specify the four actions of robots defined the casual relationship between their action and the effected status. One of precaution was to make sure the actions and the effects are made in corresponding timely manner and the changes made by the actions must be applied without contradiction.

I decided to input additional rules that makes connection between suggested output format and the other one that I created. In other words, the suggested format, 'output', omits information in several cases. In the output regarding picking up, for example, it does not have the shelf's identification. Thus, I had to create the one that consists of all the required information and connected them afterward.

Goal Statement

In the end, the quantity of product in order statement must be zero. And it also must be described that every order has some number of products in initial state. This makes program to will remove all the cases with wrong location of pickup-stations and consider only the order statement specified in the instance file.

The Law of Inertia

Additionally, the law of inertia was written for any object without robot's action to maintain the same value as immediate previous time stamp. Thus, I had to define this for all objects which have time value at the end of their state function.

Constraints

In my perspective, constraints have the most important role in that it makes the program be more useful and more practical in real world. There are two types of constraints, actions constraints and status constraints. Since there are four actions of robots there are four categories of action constraints. Regarding robot's movement, I confirmed that a robot cannot move more than one slot. They also must be located within the range of grid and cannot get out of it.

For picking up, a shelf cannot be handled by both node and robot concurrently and a robot cannot steal the shelf from other robot. Also, robot can handle only one shelf. For putting down the shelf, robots must reach to the picking station, and they cannot put it down on the highway. Delivery works similar. A robot can generate delivery action when they hold a shelf containing any product and when they are at picking station. Also, it is not possible to deliver more product than the shelf contains.

I found the necessity of state constraints only for robots and shelves. For robot's status, I focused avoiding collision by preventing them from switching their position at once or from residing the same coordination simultaneously. I also set shelf's state to reside at only one of nodes or robots at one time.

Main Results and Analysis

Types of Result Encountered

I encountered three types of results, UNSATISFIABLE, SATISFIABLE with incorrect behavior of fluent, and SATISFIABLE with correct actions. UNSATISFIABLE came to me with errors making code logically wrong. This happened especially when the naming concept were in inconsistency. SATISFIABLE with inappropriate output happened when there were not enough constraints. Therefore, I had to keep aware of errors and carefully analyze the results although I got SATISFIABLE result.

Analyze of Satisfiable Results

It seems like instance#3 and #5 have lower number of n since they have only two orders to deliver at the same picking station it requires smaller n values, while #1 has the largest number of deliveries.

N-values for each instance:

Instance Number	Minimum Time Taken
#1	13
#2	11
#3	7
#4	10
#5	6

When I implement the program with such the command *clingo test.txt simpleInstances\inst1.asp -c n=15,* I get the output as specified in *Result of each instance*

I confirmed that constraints are in good compliance. Every robot moves one slot per one turn and keep their movement within the range of grid. They pick up shelf after they reach to the shelf's location and do not take it away from other robots. Most importantly, they bring the product to the pickup station as order statement specifies.

Result of 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),move(0,1),2) occurs(object(robot,1),move(-1,0),4) occurs(object(robot,2),move(0,-1),4) occurs(object(robot,2),move(1,0),6) occurs(object(robot,1),move(0,-1),7) occurs(object(robot,2),move(1,0),7) occurs(object(robot,1),move(1,0),8) occurs(object(robot,2),move(0,-1),10) occurs(object(robot,1),move(1,0),11) occurs(object(robot,1),move(0,-1),12) occurs(object(robot,2),move(1,0),12) occurs(object(robot,2),pickup,1) occurs(object(robot,1),pickup,2) occurs(object(robot,2),pickup,8) occurs(object(robot,1),pickup,10) occurs(object(robot,2),putdown,5) occurs(object(robot,1),putdown,6) occurs(object(robot,2),deliver(1,3,4),3) occurs(object(robot,1),deliver(1,1,1),5) occurs(object(robot,2),deliver(3,4,1),11) occurs(object(robot,1),deliver(2,2,1),13)
Optimization: 13
OPTIMUM FOUND

Result of instance#2:

occurs(object(robot,1),move(0,-1),0) occurs(object(robot,2),move(0,1),0) occurs(object(robot,1),move(-1,0),2) occurs(object(robot,2),move(-1,0),4) occurs(object(robot,1),move(1,0),5) occurs(object(robot,1),move(0,1),6) occurs(object(robot,2),move(0,-1),6) occurs(object(robot,1),move(0,1),7) occurs(object(robot,1),move(-1,0),8) occurs(object(robot,2),move(1,0),8) occurs(object(robot,1),move(-1,0),9) occurs(object(robot,1),move(-1,0),10) occurs(object(robot,2),move(1,0),10) occurs(object(robot,1),move(0,-1),12) occurs(object(robot,2),move(0,-1),12) occurs(object(robot,2),pickup,2) occurs(object(robot,1),pickup,4) occurs(object(robot,2),pickup,9) occurs(object(robot,2),putdown,7) occurs(object(robot,2),deliver(1,1,1),5) occurs(object(robot,1),deliver(1,3,2),13) occurs(object(robot,2),deliver(2,2,1),13)
Optimization: 11
OPTIMUM FOUND

Result of instance#3:

occurs(object(robot,1),move(-1,0),2) occurs(object(robot,2),move(1,0),2) occurs(object(robot,1),move(0,-1),4) occurs(object(robot,2),move(-1,0),4) occurs(object(robot,1),move(0,-1),9) occurs(object(robot,2),move(1,0),9) occurs(object(robot,1),move(1,0),12) occurs(object(robot,2),move(0,-1),12) occurs(object(robot,1),pickup,8) occurs(object(robot,2),pickup,8) occurs(object(robot,1),deliver(2,4,1),10) occurs(object(robot,2),deliver(1,2,1),13)
Optimization: 7
OPTIMUM FOUND

Result of instance#4:

occurs(object(robot,1),move(-1,0),0) occurs(object(robot,2),move(0,-1),0) occurs(object(robot,1),move(-1,0),1) occurs(object(robot,1),move(0,-1),2) occurs(object(robot,2),move(-1,0),2) 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(0,1),5) occurs(object(robot,2),move(1,0),6) occurs(object(robot,2),move(-1,0),10) occurs(object(robot,1),move(-1,0),11) occurs(object(robot,2),pickup,1) occurs(object(robot,1),pickup,3) occurs(object(robot,2),pickup,7) occurs(object(robot,2),putdown,3) occurs(object(robot,1),deliver(2,2,1),6) occurs(object(robot,1),deliver(3,2,2),10) occurs(object(robot,2),deliver(1,1,1),11)
Optimization: 10
OPTIMUM FOUND

Result of instance#5:

occurs(object(robot,1),move(-1,0),2) occurs(object(robot,2),move(-1,0),2) occurs(object(robot,1),move(-1,0),3) occurs(object(robot,2),move(0,1),4) occurs(object(robot,1),move(-1,0),7) occurs(object(robot,2),move(0,-1),7) occurs(object(robot,2),pickup,3) occurs(object(robot,1),pickup,4) occurs(object(robot,2),putdown,8) occurs(object(robot,2),deliver(1,3,4),6) occurs(object(robot,1),deliver(1,1,1),8)
Optimization: 6
OPTIMUM FOUND

Conclusion (Self-Assessment)

I thought this project was an extension of module 5 of CSE 579 course. In particular, I feel that I was able to collect many ideas and principles in the block scenario where grippers move the box although I did not have a chance to set movement of grippers. Anyway, it was good opportunity to learn to solve complex reasoning problems and utilize skills gained from CSE 579.

In terms of program's performance, I got SATISFIABLE results for all instances and can see that all the five results are acceptable results in my view. They did not complete the mission in a contradictory way and followed all the designed constraints well.

The hardest part was to find errors yielding UNSATISFIABLE. It was usually caused by the code that I just wrote but I had to look over whole part. For the case of SATISFIABLE with contradiction, I put additional constraints one by one until I get reasonable result. However, I also admit I might miss some constraints to define.

Moreover, I accept the fact that there is more efficient way to approach to solve this problem. For this time, I manually defined each object, but it would be some generic ways to define them automatically. If I were to have plenty of time, I would find more creative way to define the functions.

Opportunities for Future Work

I can say that ASP is highly effective tool to solve complex problems in the world in that this project can be applied to many of the real situations. In earnest, it would be significant if I design and build a program that controls warehouse automation system as expressed in the story of project.

Furthermore, it would be nice to proceed this project with other courses related to Robotics Engineering so that I can be part of building genuine model of automated warehouse. I am confidence that I will play crucial role in such project if I am aware of logics well.

References

1) Vladimir Lifschitz. Answer Set Programming. 2009

2) Stuart Russel, Peter Norvig. Artificial Intelligence, A Modern Approach, Third Edition. 2016

Appendix

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%  GENERAL DEFINITION  %%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%%% NODE %%%
state(Nid,pair(X,Y)):- init(object(node,Nid),value(at,pair(X,Y))).
node(Nid):- init(object(node,Nid),value(at,pair(X,Y))).
node_Num(N):- N=#count{Nid:init(object(node,Nid),value(at,pair(X,Y)))}.


%%% GRID %%%
state(object(highway,Nid)):- init(object(highway,Nid),value(at,pair(X,Y))).
num_Col(num_cols):- num_cols = #count{X:init(object(node,Nid),value(at,pair(X,Y)))}.
num_Row(num_rows):- num_rows = #count{Y:init(object(node,Nid),value(at,pair(X,Y)))}.


%%% ROBOT %%%
state(object(robot,Rid),on(node,Nid),0):- state(Nid,pair(X,Y)), init(object(robot,Rid),value(at,pair(X,Y))).

robot(Rid):- init(object(robot,Rid),value(at,pair(X,Y))).

move(1,0;-1,0;0,1;0,-1).

numRobots(Num):- Num = #count{Rid:init(object(robot,Rid),value(at,pair(X,Y)))}.

{occurs(object(robot,Rid),move(X1,Y1),T):move(X1,Y1)}1:- numRobots(Num), Rid=1..Num, T=0..n-1.

state(object(robot,Rid),on(node, Moved_Nid), T+1):- state(Nid,pair(X,Y)),
state(object(robot,Rid),on(node,Nid),T), occurs(object(robot,Rid),move(X1,Y1),T), state(Moved_Nid, pair(X+X1,Y+Y1)).


%%% ORDER %%
state(Oid,object(node,Nid),product_info(Pid,PQ),0):- init(object(order,Oid),value(pickingStation,PKid)), state(object(pickingSta-
tion,PKid),object(node,Nid)), init(object(order,Oid),value(line,pair(Pid,PQ))).


%%% PICKING STATION %%%
state(object(pickingStation,PKid),object(node,Nid)) :- init(object(pickingStation,PKid),value(at,pair(X,Y))), init(ob-
ject(node,Nid),value(at,pair(X,Y))).


%%% SHELF %%%
shelf(Sid):- init(object(shelf,Sid),value(at,pair(X,Y))).
state(object(shelf,Sid),on(node,Nid),0) :- init(object(shelf,Sid),value(at,pair(X,Y))), state(Nid,pair(X,Y)).


%%% PRODUCT %%%
product(Pid):- init(object(product,Pid),value(on,pair(Sid,EA))).
state(object(product,Pid),object(shelf,Sid),with(quantity,EA),0):- init(object(product,Pid),value(on,pair(Sid,EA))).
```

```
numProducts(Num):- Num=#count{Pid:init(object(product,Pid),value(on,pair(X,Y)))}.


%%% PICKUP %%%
% Need to specifiy the shelf
{occurs(pickup(Rid,Sid),T):shelf(Sid)}1:- Rid=1..NR, numRobots(NR), T=0..n-1.
occurs(object(robot,Rid),pickup,T):- occurs(pickup(Rid,_),T).
% Effect of picking up a shelf
state(object(shelf,Sid),on(robot,Rid),T+1) :- occurs(object(robot,Rid),pickup,T), state(object(shelf,Sid),on(node,Nid),T), state(object(robot,Rid),on(node,Nid),T), R=1..NR, numRobots(NR).


%%% PUTDOWN %%%
{occurs(putdown(Rid,Sid),T):shelf(Sid)}1:- Rid=1..NR, numRobots(NR), T=0..TN,TN=n-1.
occurs(object(robot,Rid),putdown,T):-occurs(putdown(Rid,_),T).
% Effect of putting down a shelf
state(object(shelf,Sid),on(node,Nid),T+1) :- occurs(putdown(Rid,Sid),T), state(object(shelf,Sid),on(robot,Rid),T), state(object(robot,Rid),on(node,Nid),T).


%%% DELIVERY %%%
{occurs(object(robot,Rid),for(order,Oid),deliver(Sid,Pid,EA),T):state(Oid,object(node,Nid),product_info(Pid,EA),T),
state(object(product,Pid),object(shelf,Sid),with(quantity,PQ),T), EA=1..PQ}1:- Rid=1..NR, numRobots(NR), T=0..TN,TN=n-1.
occurs(object(robot,Rid),deliver(Oid,Pid,EA),T):-occurs(object(robot,Rid),for(order,Oid),deliver(Sid,Pid,EA),T).
state(Oid,object(node,Nid),product_info(Pid,EA-QQ),T+1):- occurs(object(robot,Rid),for(order,Oid),deliver(Sid,Pid,QQ),T),
state(Oid,object(node,Nid),product_info(Pid,EA),T).
state(object(product,Pid),object(shelf,Sid),with(quantity,PQ-QQ),T+1):- occurs(object(robot,R),for(order,OI),deliver(Sid,Pid,QQ),T),
state(object(product,Pid),object(shelf,Sid),with(quantity,PQ),T).


%%%%%%%%%%%%%%%%%% GOAL %%%%%%%%%%%%%%%%%%
:- not state(Oid,object(node,Nid),product_info(Pid,0),T), state(Oid,object(node,Nid),product_info(Pid,_),0),
Nid=1..NN, node_Num(NN), T=n.



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%% LAW OF INERTIA %%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

state(object(robot,Rid),on(node,Nid),T+1):- state(object(robot,Rid),on(node,Nid),T), not occurs(object(robot,Rid),move(_,_),T), T<n.
state(Oid,object(node,Nid),product_info(Pid,OQ),T+1):- state(Oid,object(node,Nid),product_info(Pid,OQ),T),
state(object(product,Pid),object(shelf,Sid),with(quantity,PQ),T), not occurs(object(robot,_),for(order,Oid),deliver(Sid,Pid,_),T), T<n.
state(object(product,Pid),object(shelf,Sid),with(quantity,PQ),T+1):- state(object(product,Pid),object(shelf,Sid),with(quantity,PQ),T),
not occurs(object(robot,_),for(order,_),deliver(Sid,Pid,_),T), T<n.
state(object(shelf,Sid),on(robot,Rid),T+1):-state(object(shelf,Sid),on(robot,Rid),T), not occurs(putdown(Rid,Sid),T), T<n.
state(object(shelf,Sid),on(node,Nid),T+1):-state(object(shelf,Sid),on(node,Nid),T), not occurs(pickup(_,Sid),T), T<n.



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%% CONSTRAINTS %%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%%%%%%%%%%%%%%%%%%% ROBOT %%%%%%%%%%%%%%%%%%%%
% Cannot move to more than 1 slot apart
:- not move(1,0;-1,0;0,1;0,-1).
:- occurs(object(robot,R),move(DX,DY),T), DX>1, DX<-1, DY>1, DY<-1.
% Within the range of grid
:- occurs(object(robot,Rid),move(X1,Y1),T), state(object(robot,Rid),on(node,Nid),T), state(Nid,pair(X,Y)), X+X1>NC, num_Col(NC).
:- occurs(object(robot,Rid),move(X1,Y1),T), state(object(robot,Rid),on(node,Nid),T), state(Nid,pair(X,Y)), Y+Y1>NR, num_Row(NR).
:- occurs(object(robot,Rid),move(X1,Y1),T), state(object(robot,Rid),on(node,Nid),T), state(Nid,pair(X,Y)), X+X1<1.
:- occurs(object(robot,Rid),move(X1,Y1),T), state(object(robot,Rid),on(node,Nid),T), state(Nid,pair(X,Y)), Y+Y1<1.
% Cannot do pickup/putdown/deliver and move concurrently
:- occurs(object(robot,Rid),move(X1,Y1),T), occurs(pickup(Rid,_),T).
:- occurs(object(robot,Rid),move(X1,Y1),T), occurs(putdown(Rid,_),T).
:- occurs(object(robot,Rid),move(X1,Y1),T), occurs(object(robot,Rid),for(order,_),deliver(_,_,_),T).
% Cannot move under other shelves if a robot holds a shelf already
:- state(object(robot,Rid),on(node,Nid),T), occurs(object(robot,Rid),move(DX,DY),T), state(Nid,pair(X,Y)), state(object(shelf,Sid),on(robot,Rid),T),
state(object(shelf,Sid_other),on(node,Nid_other),T), state(Nid_other,pair(X+DX,Y+DY)), Sid!=Sid_other, Nid!=Nid_other.
% No two actions concurrently
:- occurs(object(robot,Rid),Move,T), occurs(object(robot,Rid),Move_other,T), Move!=Move_other.


%%%%%%%%%%%%%%%%%%%% PICKUP %%%%%%%%%%%%%%%%%%%%
% No 2 shelves on the same location
:- state(object(shelf,Sid),on(node,Nid),T), state(object(shelf,Sid2),on(node,Nid),T), Sid!=Sid2.
:- state(object(shelf,Sid),on(robot,Rid),T), state(object(shelf,Sid2),on(robot,Rid),T), Sid!=Sid2.
% Cannot pick up the shelve more than 1
:- occurs(pickup(Rid,Sid),T), state(object(shelf,Sid_other),on(robot,Rid),T), Sid!=Sid_other.
% Cannot take away the shelf from other robot
:- occurs(pickup(Rid,Sid),T), state(object(shelf,Sid),on(robot,Rid_other),T), Rid!=Rid_other.
% No 2 robots pick up the same shelf
:- 2{occurs(pickup(Rid,Sid),T): robot(Rid)}, shelf(Sid).
% Can pick a shelf only on the node where the shelf reside
:- occurs(pickup(Rid,Sid),T), state(object(shelf,Sid),on(node,Nid),T), not state(object(robot,Rid),on(node,Nid),T).


%%%%%%%%%%%%%%%%%%%% PUTDOWN %%%%%%%%%%%%%%%%%%%%
% Robot can putdown only if they holds a shelf.
:- occurs(putdown(Rid,Sid),T), not state(object(shelf,Sid),on(robot,Rid),T).
% No putdown a shelf on the highway
:- occurs(putdown(Rid,Sid),T), state(object(robot,Rid),on(node,Nid),T), state(object(highway,Nid)).


%%%%%%%%%%%%%%%%%%%% DELIVERY %%%%%%%%%%%%%%%%%%%%
% Robot can do deliver only when it holds a shelves containing product
:- occurs(object(robot,Rid),for(order,Oid),deliver(Sid,Pid,_),T), state(object(product,Pid),object(shelf,Sid),with(quantity,_),T), not state(object(shelf,Sid),on(robot,Rid),T).
% Robot can do deliver only when it is at corresponding pickingstation
:- occurs(object(robot,Rid),for(order,Oid),deliver(_,Pid,_),T), state(Oid,object(node,Nid),product_info(Pid,_),T), not state(object(robot,Rid),on(node, Nid),T),
state(object(pickingStation,PKid),object(node,Nid)).
% Robot cannot do deliver more than product amount
:- occurs(object(robot,Rid),for(order,Oid),deliver(Sid,Pid,QQ),T), state(object(product,Pid),object(shelf,Sid),with(quantity,PQ),T), QQ>PQ.
% Robot cannot do deliver if order amount is larger than product amount
```

```
:- occurs(object(robot,Rid),for(order,Oid),deliver(Sid,Pid,QQ),T), state(Oid,object(node,Nid),product_info(Pid,PQ),T), QQ>PQ.


%%%%%%%%%%%%%%%%%%    ROBOT STATES    %%%%%%%%%%%%%%%%%%
% Cannot swap the location = no collision
:-    state(object(robot,Rid1),on(node,Nid),T),    state(object(robot,Rid1),on(node,Nid2),T+1),    state(object(robot,Rid2),on(node,Nid2),T),
state(object(robot,Rid2),on(node,Nid),T+1), Rid1!=Rid2.
% No 2 robots are on same location
:- 2{state(object(robot,Rid),on(node,Nid),T):node(Nid)}, robot(Rid), T=0..n.
:- 2{state(object(robot,Rid),on(node,Nid),T):robot(Rid)}, node(Nid), T=0..n.


%%%%%%%%%%%%%%%%%%    SHELF STATES    %%%%%%%%%%%%%%%%%%
% Shlef cannot be at both robot and node concurrently
:- state(object(shelf,Sid),on(node,_),T), state(object(shelf,Sid),on(robot,_),T).
:- state(object(shelf,Sid),on(node,Nid),T), state(object(shelf,Sid),on(node,Nid2),T), Nid!=Nid2.
:- state(object(shelf,Sid),on(robot,Rid),T), state(object(shelf,Sid),on(robot,Rid2),T), Rid!=Rid2.
timestamp(N):-N=#count{T:occurs(A,B,T)}.
#minimize{N: timestamp(N)}.
#show occurs/3.
```