Knowledge Representation and Reasoning
Automate Warehouse

Name: Mukesh Kumar Rohil
Email: rohil@pilani.bits-pilani.ac.in

Name: Raju Singh
Email: p20200106@pilani.bits-pilani.ac.in

Abstract

The report details out programming approach for automating warehouse using Answer Set Programming.

Problem Statement

In warehouses, there are several processes which can be automated in order to reduce the dependency on the human and introduce robots which are specifically programmed to work continuously with any supervision. One such scenario could be pickup-drop from station to station for order fulfilment purposes. Robots, specifically programmed using Answer Set Programming could freely move between different section of the warehouse based on predefined instruction.

For the matter of simplicity, we can consider warehouse to be a form of matrix, n*n. In this matrix, robots can roam around, however, the movement itself is controlled. Each robot is program to run through a definitive path, schedule the delivery, detect collision and self-recover.

Conditional Exceptions: Time is a critical factor in the warehouse. So, the time between pick and delivery is calculated for each of the robots. To enable it, we must not enter any situation where robots could collide with each other while delivering. If collision detection component must be executed, and the probability of collision must converge towards zero.

Overview of knowledge representation system.

Project Background

In any supply chain management framework, warehousing and warehouse management is a key component. A knowledge based systems, powered with an efficient programming base and algorithms would help solve key warehousing automation problem.

Answer Set Programming language is designed to provide knowledge representation based approach. It provides a robust methodologies to solve the problem. While coding the algorithm, one would be
able to key in the constrains, collision detection, source and destination.

**Approach to solve the problem.**

The problem can be solved using:
- Material collection process.
- Verification process.

First step would be to create rules which is based on behaviour driven development methodologies (BDD). In BDD, we start with generating patterns for robots behaviours, break it down into simple rules, code rules in ASP, and execute in CLINGO. As the satisfiability rules in place, next starts the integration specification for these rules in iterative manner. Integration helps so that each rule can run in its entirety, without breaking the expected output. The programmed robot can move to adjacent cells, performs all necessary actions.

**Main results and analysis.**

A n*n (4*4) cells is created. The cell or grid includes product, shelves, pickup and delivery stations and order. The goal is of the order 2,2,0,m.

Robots are initially placed at a unique location in idle state. Inputs are provided to the robots in form of order, shelves locations, pick station, delivery stations, routs to follow, detect collision etc. With the given satisfiability constraints, the code can be executed as:

Run the code:

```
$ clingo warehouse.lp instance.asp -c n=4 -c m=7
```

Format the output:

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

Snap:

Conclusion (Self-assessment)

The main purpose of designing such problem statement and find an optimal solution is to solve key bottlenecks of supply chain management, on which the whole world is dependent on. Create an insight of the projects, constraints break down constraints, create satisfiability rules, perform integration tests on the each component.

The code is modular with different sections such as: Actions, fluent, law of inertia etc.

Constraints:

- `:- not [robot(R,X,Y,D)]. robot(R,X,Y,D). T=0..m.`
- `:- not [shelf(S,X,Y,D)]. shelf(S). T=1..m.`
- `:- not [product(I,S,U,D)]. product(I). T=0..m.`
- `:- not [order(O,I,U,D)]. order(O). T=0..m.`
- `:- not [carries(R,T)]. carries(R,T). robot(R,X,Y,D). T=0..m.`. 
Figures

Step 0
Action: occurs(object(robot,1),move(-1,0),3)

Step 1
Action: occurs(object(robot,1),move(0,-1),1)

Step 2
Action: occurs(object(robot,1),move(0,1),2)
occurs(object(robot,1),pickUp,2)

Step 3
Action: occurs(object(robot,1),move(0,-1),3)

Step 4
Action: occurs(object(robot,1),move(1,0),4)

Step 5
Action: occurs(object(robot,1),move(0,-1),5)

Step 6
Action: occurs(object(robot,1),move(1,0),6),
occurs(object(robot,1),deliver(2,2,1),6)

R(1)  S(4)  P(1)  R(1)  S(4)

Robot  Shelves  Pickup Station  Picked Up
Opportunities for future work

ASP and KRR are powerful tools and concepts which goes hand in hand to solve complex real time problem. The idea is to solve problem with logic functional programming.

References


Chao and L Thomaz, 2012.


