Knowledge Representation and Reasoning (CSE579) MCS Portfolio Report

Vo Minh Duy
voduy1@asu.edu

***Abstract*—The solution in this report demonstrates the capability to perform automated reasoning about actions by using Knowledge Representation and Reasoning (KRR) tools. Clingo, which is a KRR tool, was used in this project to solve the Automated Warehouse problem. This report details the problem and challenges, the approach solution to solve the problem, and the solution's results.**

***Index Terms*—Clingo, Knowledge Representation and Reasoning, reasoning about actions, Automated Warehouse**

I. Introduction

When we perform an action, it may affect the other objects and change the state where we are in. For example, when I make an omelet, this action leads to a change in eggs' numbers. Or when I move a chair, this action leads to the change of chair's location. The chair's location or eggs' numbers are the states in this example. To convey the information about the properties of actions to computer knowledge and derive new information, we need a formal language that can encode this information and describe the effects of actions. Answer Set Programming (ASP) is a form of declarative programming that can help us in representing knowledge to computer. In ASP, the developer describes the problem, an ASP program, but does not specify the algorithm to solve the problem. The software systems that perform the search for solutions call answer set solvers, and Clingo is one of these software systems. ASP enables representing knowledge and reasoning about actions. ASP can also represent transition systems, whose states can be changed by performing actions. More importantly, ASP can help solve this type of problem, namely the planning problem, whose task is to find a sequence of actions that leads a transition system from a given initial state of the warehouse to a goal state where all orders have been fulfilled.

In this project, I solved the Automated Warehouse Scenario problem using Clingo. Automated Warehouse Scenario is an ASP Challenge Problem introduced in ASP Challenge 2019 by Martin Gebser and Philipp Obermeier [1]. In this particular problem, the problem provider introduced an automated warehouse scenario as follows:

- The warehouse is represented as a rectangular grid. Each cell on the warehouse is considered an object node with a value of a pair of (X, Y), representing the cell's coordinate. X and Y can take values in the range from smallest are 1 to the maximum value depending on the input.
- There are two types of objects on the warehouse's floor: robots and shelves. The input defines the number of robots and shelves. We can determine the robot and shelf's location by the cell coordinate that hosts the object at a given time. The robots can pick up and deliver products from shelves to picking stations based on defined orders. While robots can move horizontally or vertically between adjacent cells underneath shelves, pick up a shelf, and put down a shelf, if a robot carries a shelf, it can not move freely underneath shelves anymore. Therefore, the program may need to make a path to reach and bring out the objective shelf.
- There are highway cells on the warehouse grid where the robot can move while carrying a shelf but can not put down the shelf on such cells. The input defines the location of highway cells.
- Problem's input also provides information about products and orders. Information about the product included the product's location, i.e., which shelf hosts the product and its quantities on that shelf. One product may be placed on multiple shelves. Each order consists of the required product, their quantities, and the picking station to collect them. When a robot carries the shelf that hosts the required product to the order's picking station, the robot can perform delivery action, which reduces the product's quantity on the shelf and in the order by a number U. This considers U units of the product have been delivered. But neither of the resulting quantities of reducing action may be negative.

In the scenario, the goal is to fulfill the orders. By a given input, the program can automatically produce a plan, which represents the robot's movement to fulfill the orders under the above constraints.

II. Explanation of solution

A. Knowledge preparation for the project

Before begin developing the program, the course's project milestones one and two help me get familiar with ASP and Clingo. I studied the theory ASP and a few ASP practices in weeks three and four, including the graph problems, sudoku puzzles, n-queen puzzles. In week five, the knowledge on reasoning about actions contributes significantly to the background knowledge in helping me on solving this project's problem. Mainly, the knowledge about the concept of transition system and expressive probabilities, reasoning about actions in ASP. Additionally, the graded assignments in week five and practicing more on the block world problem before beginning the project also provided familiarity and help in understanding the concept. These works build a solid knowledge to begin solving the automated warehouse problem.

ASP is an approach to knowledge representation and reasoning. In this program, the knowledge about the warehouse state, including objects like robots, shelves, and products, robots' possible actions, is represented as answer set program. The reasoning is performed by Clingo, and new information is derived from this encoded knowledge. The plan to fulfill orders is the desired information which is derived based on input knowledge and the required goal. Go deeper into this scenario, to describe sequentially performed robot' actions, we need to represent the time concept, in other words, the steps that the state may transition from one to another after robots perform an action. The time instants for presenting the robot status, actions, orders status, and product status are denoted by integers between 0 and a non-negative integer step that the user input when executing the program.

One more important thing to note is the required ability to express commonsense reasoning in this transition system. If the robot is carrying a shelf, it will remain carrying if no put-down action is performed. The shelf's location will remain the same if no robot picks it up and move to another cell. Because ASP enables default reasoning, we can presume that the fluent has the same value as before if there is no evidence to the contrary. For example, in the program, the following three rules together represent the default reasoning about robot status. By default, rule (1) represents the robot status is free in the next step if it is free in the current step and no pick-up action is performed. Rule (2) says that if the robot carries a shelf in the current step, it will carry that shelf in the next step if no put-down action is performed. Rule (3) states that the robot is either free or carries a shelf at each time step. In these rules, T represents the time concept in the program.

$$\begin{aligned}&\{free(object(robot,RB),T+1)\}:-\\&\qquad free(object(robot,RB),T),T<step.\end{aligned}\tag{1}$$

$$\begin{aligned}&\{carrying(object(robot,RB),object(shelf,S),T+1)\}:-\\&\qquad carrying(object(robot,RB),object(shelf,S),T),T<step.\end{aligned}\tag{2}$$

$$\begin{aligned}&:-\,not\ 1\{free(object(robot,RB),T);carrying(object(robot,RB),\\&\qquad object(shelf,S),T)\}1,object(robot,RB),T=1..step.\end{aligned}\tag{3}$$

B. Approach to solution

The automated warehouse scenario is complex, involves many parties take part in it. For each state of the system, we need to consider the state of many objects like products' quantity on shelves, shelves' location, robots' location and executed actions, products' quantity in orders. Additionally, we also need to consider the constraint placed on each object and preconditions for actions. Implementing all the objects, constraints, actions at once will be very difficult to track the tasks and progress. Additionally, it will be challenging to check whether the program is not working correctly or simply the problem is unsatisfiable. Even if we know the program is not working correctly, track down which rule is the cause will not be an easy task.

My approach to working toward the solution is dividing the system into many small parts that I can solve and test sequentially. This approach helps efficiently solve a particular action or constraint, track the tasks and check for the correctness of the implemented part. By dividing the program into small parts, when adding new action or object, the program can be easily tested with the defined goal for the correctness and check if there is any impact on the last implemented parts.

Started from representing the initial input, I declared new atoms to represent individual objects like robots and shelves, new atoms to represent robots and shelves location at each time step. Also, based on initial input, the location of robots and shelves at time step 0 is declared. At first, I consider the narrower problem, which consists of only one robot and movement action. The goal is for the robot to move correctly to the desired coordinate on the grid. I implemented the effect of move action as it changes the robot's location; the constraint is the location after move action must be a valid coordinate. At this step, the goal of the problem is for the robot to move to the desired cell, which is the constraint I set in the goal section. One detail that needs to be considered is the range of robot movement. To restrain the movement range only at one cell horizontally or vertically between adjacent cells, I define values of DX, DY as follows: $DX=-1..1, DY=-1..1, |DX+DY|==1.$. With this definition, the possible value of DX, DY can only be (-1,0), (1,0), (0,-1) or (0,1). These declarations can make sure that the robot only moves one cell horizontally or vertically at each step.

Next, I added the pick-up action together with its effects and constraints. Because the robot's movement is constrained when carrying a shelf, I added new atoms to represent the robot's status at each time step: $free$ and $carrying$. When the robot is free, it can move freely and perform pick-up; when the robot is carrying a shelf, it can not move to another cell that is hosting a shelf, and it can not perform pick-up action anymore. At time step 0, all robots are free. Furthermore, the robot status is constrained to either free or carrying at each time step. Rule (3) in the previous section demonstrates the uniqueness and existence of the robot's status at each time step. The $carrying$ atom also represents the following knowledge: which robot carries which shelf at a particular time step. One more point to be noted is when the robot's status is carrying, then the carried shelf's location is derived from the robot's location. Rule (4) represents this knowledge. In the program, the atom $robot\_loc$ and $shelf\_loc$ define the robots and shelves' location. The rule states that if a robot carries a shelf, then the robot's location implies the shelf's location.

$$\begin{aligned}&shelf\_loc(object(shelf,S),value(at,pair(X,Y)),T):-\\&\qquad carrying(object(robot,RB),object(shelf,S),T),\\&\qquad object(robot,RB),object(shelf,S),\\&\qquad robot\_loc(object(robot,RB),value(at,pair(X,Y)),T).\end{aligned}\tag{4}$$

Next, I move to implement the put-down action. Put-down action is opposite to pick-up action; it changes the robot's status from carrying to free. When the shelf is put down, its location will no change, and by default, it will be the same in subsequence steps; this is achieved by rule (1). The precondition is when the robot puts down the shelf, the cell that hosts the robot at the previous step must not be the highway cell. For testing, the goal is shelf's location must be the same as the picking station's coordinate, and the robot is free at the given time step.

All the above works provide a base system for the automated warehouse. My subsequent implementations are the addition of orders, products, and deliver action. Consider that there is the need to track and represent products' quantities on the shelf and products' quantities in order, I use two new atoms to represent this information: $inventory$ and $order\_fullfill$. The atom $inventory$ represents product and shelf relation, product' quantities (5). The atom $order\_fullfill$ represents the relation between order and the required product, include the required quantities of the product (6). With these two atoms, the program can track the product's quantities at each time step.

$$order\_fullfill(object(order, O), value(line, pair(P, U)), T) \quad (5)$$
$$inventory(object(product, P), value(on, pair(S, U)), T) \quad (6)$$

The deliver action is declared as follows: the robot must be at the order's picking station and is carrying a shelf, and the shelf must contain the product need by the order. Up until this point, to reduce the complexity, the program works with only one robot. Next, I added more robots and constraints to prevent robots' collisions. I use two rules for preventing collisions: two robots can not be on the same cell simultaneously, and two robots can not switch cells to each other at the same time. With all the parts of the program being integrated, the last task is testing the program with the provided simple instances and verifying the program's result. The goal is all orders must be fulfilled, meaning that the product's quantities must be 0 at the given time. Rule (7) constraint the goal state for the problem:

$$\begin{aligned} :- \; & not\ order\_fullfill(object(order, O), value(line, pair(P, 0)), \\ & step), init(object(order, O), value(line, pair(P, U))). \end{aligned} \quad (7)$$

III. Results

The program is implemented using the approach described in section two and executed with the test instances that the problem provider provides. For each provided test instance, I execute the program with an input time step range from 1 until the stable models can be found. The achieved results are as in table 1. In table 1, the minimum step is the lowest step at which the program finds the plans. The number of plans is the total plans that the program found at the corresponding step.

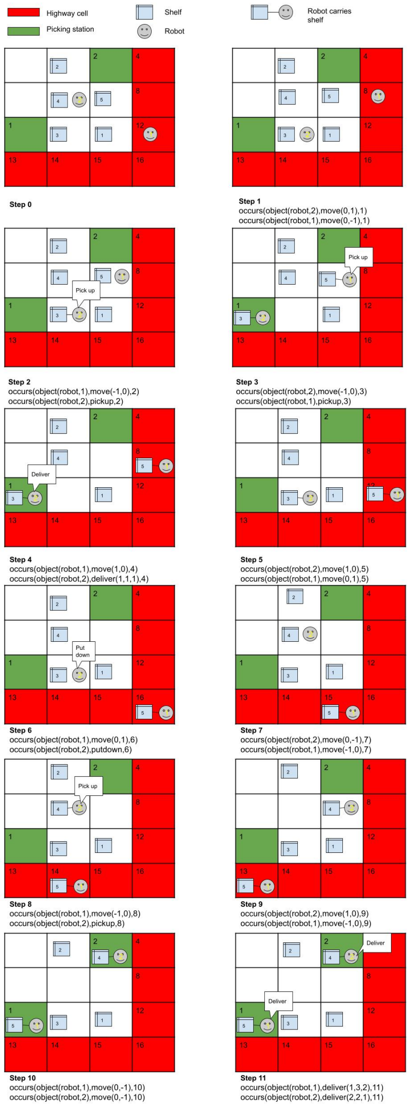
Based on these results, I can say that the implemented solution, which is using ASP and Clingo, can solve the

TABLE I
Results of program tested with provided instances.

	Minimum step	Number of plans
Simple instance 1	13	24
Simple instance 2	11	6
Simple instance 3	7	136
Simple instance 4	10	1348
Simple instance 5	6	20

automated warehouse problem stated in the ASP Challenge 2019. Not only with the provided simple instances, with given instances of the problem where the grid size, robot numbers, products, and orders are arbitrary numbers, the program can be expected to find and produce a plan to fulfill the requirements given sufficient time steps.

Fig. 1. Visualizing a result plan for simple instance 4



Step 0

Step 1
occurs(object(robot,2),move(0,1),1)
occurs(object(robot,1),move(0,-1),1)

Step 2
occurs(object(robot,1),move(-1,0),2)
occurs(object(robot,2),pickup,2)

Step 3
occurs(object(robot,2),move(-1,0),3)
occurs(object(robot,1),pickup,3)

Step 4
occurs(object(robot,1),move(1,0),4)
occurs(object(robot,2),deliver(1,1,1),4)

Step 5
occurs(object(robot,2),move(1,0),5)
occurs(object(robot,1),move(0,1),5)

Step 6
occurs(object(robot,1),move(0,1),6)
occurs(object(robot,2),putdown,6)

Step 7
occurs(object(robot,2),move(0,-1),7)
occurs(object(robot,1),move(-1,0),7)

Step 8
occurs(object(robot,1),move(-1,0),8)
occurs(object(robot,2),pickup,8)

Step 9
occurs(object(robot,2),move(1,0),9)
occurs(object(robot,1),move(-1,0),9)

Step 10
occurs(object(robot,1),move(0,-1),10)
occurs(object(robot,2),move(0,-1),10)

Step 11
occurs(object(robot,1),deliver(1,3,2),11)
occurs(object(robot,2),deliver(2,2,1),11)

Figure 1 visualizes an example result found for test instance two. In test instance two, order one requires products one and three delivering to picking station one, order two requires product two delivering to picking station two. Product one is on shelf three, and product two is on shelf four. Shelf five, which hosts product three, is surrounded by shelves one, four, and five. To deliver product three to picking station one, robot two will pick up shelf five and move on the highway to the destination. While robot two delivers product three, robot one delivers products one and two to the picking station. The plan ASP program created is precise and very similar to how humans thinking. More impressively, it takes a short time for the ASP solver to search for the answers. Even though it is just a scenario in a narrow domain, it is fascinating to see this kind of reasoning in the machine.

IV. Lessons learned

Throughout this project, I gained significant knowledge of the ASP language and new programming skills with Clingo. Firstly, with project milestones one and two, I gained experience in programming ASP with Clingo. Especially in week four ”Simple Answer Set Programming” assignment, solving n-queen puzzles, graph problems, and sudoku puzzles help me build a solid understanding of ASP and Clingo. I can understand and use choice rules to generate desired search space in the problem. I also can construct constraints to weed out undesired stable models. The main contribution to completing the project is the knowledge in week five. In this week, I gained knowledge of reasoning about actions in ASP. I understand the concept of the transition system, time concept, and law of inertia in ASP. With this knowledge and practice on week five’s assignment, I know how to represent an occurrence of action and define default in ASP. As a result, the program I developed by Clingo can perform automated reasoning about actions and solve the Automated Warehouse problem.

The knowledge I gained is not only from the materials provided in the course. While researched for more understanding of ASP, I came across other materials that the authors explain about the application of ASP in industrial domains. There are other applications of ASP, such as assembly planning [2], Decision support system for the Space Shuttle [3]. In the tourism industry, there is an application in which an ASP-based software is integrated into an e-tourism portal [4]. All the materials helped me to realize that there are broad applications for ASP in solving real-world problems.

References

[1] M. Gebser and P. Obermeier, “Asp challenge problem: Automated warehouse scenario,” Available at https://sites.google.com/view/aspcomp2019/home?authuser=0 (2019).

[2] M. Rizwan, V. Patoglu, and E. Erdem, “Human robot collaborative assembly planning: An answer set programming approach,” *CoRR*, vol. abs/2008.03496, 2020. [Online]. Available: https://arxiv.org/abs/2008.03496

[3] M. Nogueira, M. Balduccini, M. Gelfond, R. Watson, and M. Barry, “An a-prolog decision support system for the space shuttle,” in *Practical Aspects of Declarative Languages, Third International Symposium, PADL 2001, Las Vegas, Nevada, USA, March 11-12, 2001, Proceedings*, ser. Lecture Notes in Computer Science, I. V. Ramakrishnan, Ed., vol. 1990. Springer, 2001, pp. 169–183. [Online]. Available: https://doi.org/10.1007/3-540-45241-9_12

[4] F. Ricca, A. Dimasi, G. Grasso, S. M. Ielpa, S. Iiritano, M. Manna, and N. Leone, “A logic-based system for e-tourism,” *Fundam. Informaticae*, vol. 105, no. 1-2, pp. 35–55, 2010. [Online]. Available: https://doi.org/10.3233/FI-2010-357