CSE 579: Knowledge Representation and Reasoning MCS Project Portfolio Report

1st Andy Gonzalez
School of Computing and Augmented Intelligence
Arizona State University
Tempe, Arizona
agonz600@asu.edu

***Abstract*—This report summarizes the final project for CSE 579: Knowledge Representation and Reasoning for inclusion in the MCS portfolio. In this report, we aim to provide an answer set programming (ASP) solution to robot warehouse automation scenario in which we simulate a warehouse filled with robots and find the most efficient way possible to fulfill orders of items. This report includes a introduction in which we describe the problem statement, an overview of background knowledge required to tackle this problem, a description of the solution, a description of the results, a statement of contributions, and a personal self reflection.**

***Index Terms*—Answer set programming, Clingo, Propositional logic, First-order logic, Knowledge representation**

I. INTRODUCTION

The problem scenario is titled ”Robot Automation Warehouse Scenario.” This problem was a part of ASP Challenge 2019 and can be found on their website [3].

A. Problem Statement

In a warehouse represented as a grid, we have many flat robots that can move under shelves freely. However, once a robot carries a shelf it cannot move into spaces that contain shelves and shelves cannot be placed on warehouse highways. The goal of robot is to deliver products on shelves to picking station so that orders can be fulfilled. Orders are fulfilled once the product in the order is fulfilled. This warehouse must be as efficient as possible.

B. Project Background

In this section, we detail concepts that are necessary to understand in order to solve this problem using knowledge representation and reasoning (KRR) principles.

a) Propositional Logic: Propositional Logic (PL) is a field of mathematical logic that concerns itself with combining true and false statements, called propositions, with propositional connectives: conjunction, disjunction, implication, equivalence, and negation. There are a few limitations to PL [6]:

- It cannot handle quantifiers: exists, all, some, none.
- It cannot handle relationships between individuals and objects.

To solve these, we use first-order logic.

b) First Order Logic: First-order logic (FOL) we say is more expressive than PL. It introduces objects which represent individuals, predicates which represent properties of individuals, and functions which represent mappings between objects. It also introduces the universal and existential quantifiers, $\forall$ and $\exists$, to represent statements ‘for all’ and ‘there exists’. Lastly, FOL introduces the concept of variables, that serve as place holders for real values. These new concepts introduced in FOL allow for more expressive representation of statements that were not possible under PL.

c) Satisfiability: We say an interpretation I, is a model of a set of logical formulas, Γ, if I satisfies every formula, F, in Γ. We say that Γ is satisifiable if there exists an I that satisfies every formula in Γ [3, pp. 5]. Informally, I is the set of objects, predicates, and functions from the signature of Γ which are true.

Finding all models of Γ is an NP-hard problem [4, pp. 21], meaning the only general solution is to enumerate all possible interpretations and check if they are models. To do this efficiently we use answer set solvers and a programming language called ASP.

d) Answer Set Programming: ASP is a logical programming language based on FOL. Typically, ASP programs need grounders to convert FOL to Herbrand FOL. Herbrand is simply the grounding of all terms in a FOL program, which means that every rule is variable-free and is interpreted as itself. ASP also needs an answer set solver to find stable models, also known as answer sets [2, pp. 45-47]. A stable model is a minimal model, meaning there is no other model that is a smaller subset of the stable model.

Semantics of ASP. In ASP, a predicates encompasses both predicates and functions of FOL. An atom is an instance of a predicate. The arity of a predicate indicates the number of arguments the predicate takes. The symbols $:-$, indicate an implication where the body in the right-hand side is implying the head on the left-hand side. A stable model must satisfy all rules.

e) Clingo: In this project, we use Clingo. Clingo combines Gringo and Clasp. Gringo is a grounder and Clasp is an answer set solver. Clingo was installed through the Python distribution Anaconda following [1]. The syntax and semantics of Gringo were found in [5].

II. Solution

The general method of solving a problem using KRR principles is to represent the objects, define the initial states of the objects, precisely represent the actions of objects and the effects of said actions, define constraints that limit things that cannot happen in the real-world, and define constraints to ensure that the intended goal is achieved. All these tasks are relatively simple and detailed in this section. Note that in Clingo, there are no traditional objects as there are in traditional programming languages. When we refer to objects, we refer to predicates with an arity of one that represent objects in the real world.

To solve this problem, we developed the main program iteratively. We divided the problem into multiple components:

- Input Processing: Handling how we receive the initial state of all objects and orders to be fulfilled.
- Robot Action Logic: Handling how a robot moves, the preconditions and effects of its actions.
- Order Fulfillment Logic: Handling how an order is updated and fulfilled.
- Output Processing: Reporting the details of the actions taken.

A. Input Processing

The input to this problem and program is of the following form:

```
init(object(node,1),value(at,pair(1,1))).
```

Where $init$ is a function that stands for initial state. It has two arguments, the object and the value. The possible objects for this problem were: $node$, $highway$, $robot$, $pickingStation$, $item$, and $product$. There were three value atoms with different meanings: *at*: indicates initial position, *on*: indicates an amount of product on certain shelf, and $line$: indicates a product and amount of product required by an order.

To process the input, we create objects for all the input objects, and a corresponding function for all the values. This is how a $robot$ object is created.

```
robot(R) :- init(object(robot,R),_).
```

Objects were created for all input objects except for the $node$ objects. Instead, a single $grid$ object was created spanning all grid values. Another important object are $item$ objects, these are parts of orders that indicate the order, the product, and the quantity of the product.

Another initialization task was defining a $time$ predicate that ranges from 1 to a defined constant *steps*. The purpose of $time$ is to mimic time in the real-world, and to track when robots take actions. Lastly, we define an $amount$ predicate that ranges from 0 to a defined constant max_amount. The purpose of $amount$ is to ground rules in which the quantity of products appears in the head of a rule and not in the body, and it represents the valid range of the quantity of products.

B. Robot Action Logic

When writing ASP rules, it is important to define the preconditions and effects of every action. On every time step, the robot could perform the following actions:

a) $Move$: First, we defined a predicate called $Move(Dx, Dy)$, where Dx and Dy span the values $[-1, 0, 1]$, where the sum of both absolute values is at most one. This represents the four possible moves to the four neighboring cells (locations on the grid). The precondition for moving a robot at position (X, Y) is that $(X + Dx, Y + Dy)$, must be inside the defined grid. This is represented by the following rule:

```
{movement(robot,R,Dx,Dy,T)}:-
at(robot,R,X,Y,T),
robot(R),
grid(X+Dx, Y+Dy),
move(Dx, Dy), time(T).
```

This rule can be read as "if a robot R is at (X, Y) at time T and it can move to cell at $(X + Dx, Y + Dy)$ inside the grid, then it may or may not move to the cell at time T." Note, the choice rules, $\{\}$, allows for the Clingo to consider both possibilities of movement and non-movement

The effect of moving a robot to a new position is that the robot is at the new position at time $T + 1$. Another effect is if the robot is moves while carrying a shelf, the shelf will also be at the robot's new position at time $T + 1$. The constraints on movement are that a robot cannot move to another cell if the cell is occupied by another robot, a robot carrying a shelf cannot move to another cell if the cell is occupied by another shelf.

b) $Pickup$.: The precondition of picking up a shelf at time T is that the robot and the shelf both must be in the same cell, and the robot cannot be carrying another shelf at time T. The effect of picking up a shelf at time T is that the robot is now carrying shelf at time $T + 1$. The effect rule is represented by the following rule:

```
carrying(R, S, T+1) :- pickup(R,S,T),
time(T).
```

The scenarios such as a robot picking up a two shelves at a time, or picking up a shelf that is already being carried, cannot occur as those are constraints are entailed by the constraints on robot movement.

c) $Putdown$.: The precondition for putting down a shelf at time T is that the robot must be carrying a shelf at time T. The effect of putting down a shelf at time T is that the robot is no longer carrying a shelf at time $T + 1$. A constraint on putting down a shelf is that a robot cannot put down a shelf if they are on a $highway$ cell. This constraint is exemplified by the following rule:

```
:- at(highway, H, X, Y, T),
putdown(R, S, T),
at(robot, R, X, Y, T),
time(T).
```

Constraints do not have heads in ASP, meaning if the body is true, then the rule is not satisfied. These types of rules effectively filter out undesirable stable models.

d) Deliver.: The precondition for delivering an item, is that the robot must be carrying a shelf with the necessary product to fulfill the item's required product at time T, and the robot is at the picking station for the order to which the item belongs at time T. Furthermore, the amount delivered cannot exceed the amount on the shelf nor the amount requested, and the amount delivered cannot be zero. The effect of delivering an item, is that the amount of product delivered is subtracted from the amount of product on the shelf and the amount of product requested by the item.

The overarching constraints on all actions are that a single robot can only do a one action per time-step.

C. Order Fulfillment Logic

An order in this scenario is considered fulfilled, once all amounts of order items are at 0 at time $steps + 1$. This rule is a constraint:

```
:- not item(O, I, 0, steps+1),
item(O,I,U,T), time(T).
```

The optimal solution is found by finding the minimal time-step for which the program is satisfiable. There were no rules included in the program that specifically automated this process. Rather, the optimal solution was found by manually searching for the minimal time-step by adjusting the constant $steps$ in the command line when running the program.

D. Output Processing

The output for this problem needed to follow the following format:

```
occurs(object(robot,R),move(Dx,Dy),T)
```

The predicate $occurs$ has an arity of 3. The first input is the an $object$ predicate with an arity of 2 with its first input being the word "robot" and the second being the ID of the robot that caused the action. The second input is another predicate that indicates the action taken. The third input is the time-step in which the action occurred.

To generate the output, every action implied an $occurs$. We did this by creating rules in which the head is the $occurs$ atom with the appropriate action type, and the body is the action the robot took at time T. Essentially, this was refactoring the actions atoms that were created as part of the robot action logic, which already contained all the required information, into the expected output for this scenario.

III. Results

To analyze the efficacy of the resulting program, the program is ran it against five scenarios given as part of the 2019 ASP Challenge [3]. Each scenario is a $4X4$ grid with highways along the bottom-most and right-most column of the grid. Scenarios include two robots located at $grid(4,3)$ and $grid(2,2)$, two picking stations located at $grid(1,3)$ and $grid(3,1)$, six shelves located at $grid(3,3)$, $grid(2,1)$, $grid(2,3)$, $grid(2,2)$, $grid(3,2)$, and $grid(1,2)$, and four products located on shelf with quantity $(3,1)$,$(4,1)$, $(6,4)$, and $((5,1)\ (6,1))$. In the scenarios, when only a subset of these atoms has been specified, they are included in the order listed.

- The first scenario is solved in 13 steps. It contained three orders requesting product-quantity $((1,1),\ (3,4))$, $(2,1)$, and $(4,1)$ respectively.
- The second scenario is solved in 11 steps. It contained three products, five shelves, and two orders requesting product-quantity $((1,1),\ (3,1))$ and $(2,1)$ respectively.
- The third scenario is solved in 7 steps. It contained one picking station, two orders requesting product-quantity $(2,1)$ and $(4,1)$ respectively.
- The fourth scenario is solved in 9 steps. It contained two products and three orders requesting product-quantity $(1,1)$, $(2,1)$ and $(2,2)$ respectively.
- The fifth scenario is solved in 6 steps. It contained one picking station, and one order requesting product-quantity $(1,1)$ and $(3,4)$.

The solution for the first problem is given by [3], and we verified that our solution aligned.

We can also verify the efficacy of the program manually. To further analyze the results of the program let us consider a solution for the fifth scenario:

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

Tracing the steps. Robot one moves down twice to $grid(2,3)$ at time $T = 2$, it performs no action at $T = 3$, performs a pickup action at $T = 4$, moves one down to the picking station at $T = 5$, and delivers at $T = 6$. Robot two moves down to $grid(1,2)$, pick ups a shelf at $T = 2$, moves right one to the picking station at $T = 3$, delivers at $T = 4$, and moves one left at $T = 5$. We can see that robot two immediately delivered and moved out of the way so that robot one could deliver, satisfying the program in six time steps.

This move set is optimal in that it is completed in the least amount of possible time steps. We can envision optimal alternatives in which the robot one pickups the shelf and moves to deliver immediately, then has to move out of the way for robot two, or the robot two moves out of the way in a different direction. Scenarios can have hundreds or thousands of stable models. Potentially, we could have used one of Clingo's optimization functions to find the solution with the least number of actions taken, a truly optimal solution. This could be done by minimizing the occurrences of the $occurs$ atom. However, in this problem we defined optimally solely based on the number of time steps taken in total.

The results showcase the applicability of Clingo and ASP in representing and solving real-world problems. Results to all the scenarios were delivered in less than a second for a single solution, but to list all solutions Clingo can take a few seconds to few minutes depending on the scenario. Clingo can effectively be used to solve NP-hard satisfiability problems that occur in FOL through a combination of using grounders and answer set solvers to find the stable models of such programs.

IV. Contributions

This project was completed individually. To complete this project I learned: the fundamentals of logic paradigms such as PL, FOL, and second order logic (SOL) and its role in representation, how to program in ASP Clingo following best practices, the syntax of Gringo, and how to represent the state of the world and the effects actions have on it in Clingo. Tasks I completed for this projected were: learning background knowledge, download and installation of required programs, writing the robot warehouse ASP script, testing the ASP script, and writing the report summarizing the solution and results of the robot warehouse problem and any background knowledge I used in the development of the solution.

V. Self-Reflection

Through the development of this individual project, I took many different approaches initially. These approaches led to errors such as robots being in multiple places despite having rules to limit such occurrences. This was due to my lack of understanding of how Clingo handles the concept of negation as failure and grounding. To solve this I would re-read [2] and [5], focusing the section that concerned my problem.

I am content with the result as it works as envisioned. The ASP code itself, if written by a more advanced ASP programmer, could be written more efficiently and precisely with the necessary expertise. As my attempts for conciseness in code would have unintended side effects and therefore be removed. One specific instance I can cite was limiting the number of concurrent actions to a single action. In my program, I wrote rules for each possible pair of actions, while doing so I was sure that there was a better way to represent actions such that those multiple lines of code could be reduced to one.

I learned how to represent real-world object: robots, shelves, products, space, etc. in FOL and then translate into ASP. I learned how to abstract the relationships of these objects over time. I also learned how to think in terms of actions and effects of actions, how to write hard constraints on rules to limit when actions take place and filter which models are possible in alignment with the real-world view. Lastly, I learn how to appropriately structure ASP programs according to common ASP coding principles.

References

[1] "clingo and gringo," potassco.org. https://potassco.org/clingo/.

[2] V. Lifschitz, "Answer set programming and plan generation," Artificial Intelligence, vol. 138, no. 1–2, pp. 39–54, Jun. 2002, doi: https://doi.org/10.1016/s0004-3702(02)00186-8.

[3] "ASP Challenge 2019 - Problem Domains," sites.google.com. https://sites.google.com/view/aspcomp2019/problem-domains.

[4] Frank van Harmelen, V. Lifschitz, and B. Porter, Handbook of Knowledge Representation. Elsevier, 2008.

[5] M. Gebser, A. Harrison, R. Kaminski, V. Lifschitz, and T. Schaub, "Abstract Gringo," Theory and Practice of Logic Programming, Jan. 2003, Accessed: Jul. 13, 2024. [Online]. Available: https://www.researchgate.net/publication/280329493_Abstract_Gringo

[6] J. Lee, "Introduction to First Order Logic," lecture slides, CSE 579: Knowledge Representation and Reasoning, Arizona State University, Tempe, AZ, USA, May 20, 2024.