Knowledge Representation and Reasoning

Pathfinding in the Automated Warehouse

James William Dunn
jwdunn2@asu.edu

I. Introduction

With global commerce, there exists a continual flow of order placement and fulfillment. Products need to move from point A to point B. This report presents an approach toward the design of a micro-scale sixteen-cell solver utilizing answer set programming to achieve logically correct plans with non-conflicting paths. The modern warehouse employs a degree of automation which entails an increase in productivity, safety, and order completion accuracy; and potentially a decrease in facility and labor costs. One method to mechanize product fulfillment involves robotically moving shelving units from their initial storage positions to a target where orders are assembled. This requires planning collision-free paths for each shelf subject to a set of constraints. Consider a simplified model of a warehouse on a 4x4 grid. The aim is to plan the activity of two robots with several virtual shelves and some example orders. Here, the allowable actions would be movement from cell to cell, traveling beneath shelves, picking up or putting down a shelf, and delivering products from the shelves to a target picking station. This rudimentary world can be represented with formulas in a logic program.

II. Solution

An example configuration of a simulated warehouse is presented in Fig. 1. The robots are indicated as white squares and the shelves as orange circles. The picking stations are striped and the purple cells signify a highway where shelves may not be placed. To represent this virtual world in the form of declarative rules, the transition framework in the blocks world domain described by Lifschitz [1] provides an infrastructure which can be extended to a robot grid. The fundamentals of automated reasoning and pathfinding in the answer set programming domain [2] and applications in robotic logistics [3, 4] serve as additional background.

To reason about actions in a dynamic environment requires a set of rules that define a state transition system [5]. As time progresses, objects should persist in their current location and robots carrying shelves should continue to carry them. This is the notion of a *fluent*. In the next time step, a fluent should maintain its state, unless affected by an event. This persistence of state is also known as the common sense law of inertia. Of equivalent importance is the maintenance of the uniqueness of an object. It would not make sense if our robots replicate unexpectedly through time. This is crucial within the context of answer set programming where an oversight can lead to combinatorial expansion.

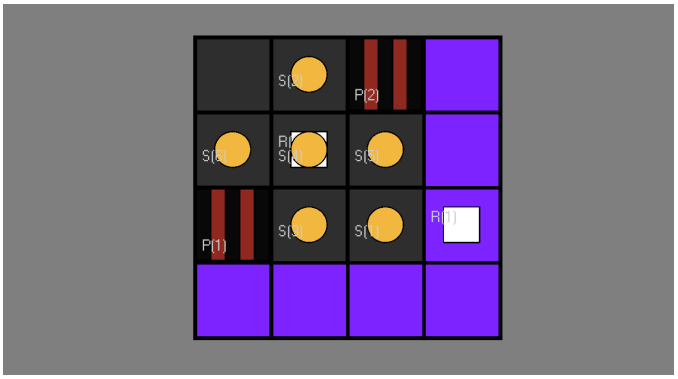

Figure 1: Visualization of a minimal warehouse in Asprilo[1].

Referring to the minimal model presented above, there are several objects and fluents that require implementation. This modeling can be constructed through four phases of development. The initial phase establishes a three-cell robot grid. This prototype models a microcosm where the goal is to move a robot from cell 1 to cell 2, pick up the shelf in that cell, and finally deliver it to cell 3. Within this context, an action domain with an initial rule-base is founded on the blocks world architecture. This initial phase serves as a basis for the next three increments toward a full solution.

Phase two adds the second dimension such that two robots coexist and avoid collisions. A mini 3x3 grid is constructed and a corresponding initial arrangement is designed for testing purposes. Two shelves and two target cells are introduced to investigate robot cooperation and overlapping paths. For example, a logical constraint to prevent collision is:

```
:- botLoc(B1,X,Y,T), botLoc(B2,X,Y,T), B1!=B2.
```

In phase three, picking stations and designated highway nodes are added to a midi-sized 4x4 grid. This expansion requires an additional constraint to prevent placing shelves on a highway.

The final phase defines orders and products on the shelves. This includes computing the delivery quantities when the robots fulfill orders at a picking station. At this main level of operation, the original scenario description[2] prescribes five predefined test configurations. In some instances, a shelf may contain one part of an order while another shelf contains the balance. In another case, one shelf satisfies two orders at one picking station.

1 asprilo.github.io/visualizer
2 sites.google.com/view/aspcomp2019/problem-domains

Robots are not required to put down a shelf at a picking station – they deliver the product while carrying the shelf and afterwards may move the shelf away.

```
01  shelf(5). prod(1,5,25,0). carry(5,1,0).
02  order(1,1,9,0,0). order(2,1,16,0,0).
03  bot(1). time(1..2). item(1,1). item(2,1).
04  :- not {prod(I,S,_,T)}=1, prod(I,S,_,T-1), time(T).
05  :- not {order(O,I,_,_,T)}=1, item(O,I), time(T).
06  :- not {carry(S,_,T)}=1, shelf(S), time(T).
07  {order(O,I,U,Q,T)} :- order(O,I,U,Q,T-1), time(T).
08  {carry(S,B,T)} :- carry(S,B,T-1), time(T).
09  {deliver(O,I,U,T)}1
10   :- bot(B), carry(S,B,T-1), prod(I,S,_,T-1),
11      U=Ou-Qu, order(O,I,Ou,Qu,T-1), time(T).
12  order(O,I,Ou,U+Q,T)
13   :- deliver(O,I,U,T),
14      order(O,I,Ou,Q,T-1).
15  prod(I,S,Pu-Ou,T)
16   :- deliver(O,I,Ou,T),
17      carry(S,B,T-1), prod(I,S,Pu,T-1).
18  :- order(O,I,U,Q,T), Q!=U, T=2. #show deliver/4.
```

Figure 2: Example clingo encoding for a delivery operation.

A clingo[3] encoding is a declarative set of rules with a basic form of *head :- body*. A rule without a head is a constraint, while a rule without a body is a fact. A head may be wrapped in braces to indicate a choice rule which means that clingo may select any subset of the atoms listed, with an optional upper/lower bound to specify the required number of elements to be added to the set.

A delivery operation can be completed when a robot reaches a target picking station while carrying a shelf with products that match existing open orders. This case is described in Fig. 2. where shelf 5 contains 25 units of product 1, and two open orders require 9 and 16 units respectively. The code includes the inertia and uniqueness axioms (lines 4-8). The goal is declared on line 18, stating that there should not be any orders at time step 2 where the third and fourth parameters are unequal. For this example listing, clingo produces two logically valid answer sets: *{deliver(1,1,9,1), deliver(2,1,16,2)}* and the reverse order fulfillment sequence *{deliver(2,1,16,1), deliver(1,1,9,2)}*.

III. Results

The initial phase modeled a 3x1 grid and the robot performed its expected four steps consisting of: move, pickup, move again, and putdown. With this successful implementation of the basic operational construct, the foundational structure was actualized for further development.

The second phase modeled a 3x3 grid with two robots and two shelves, establishing the next dimension of movement within a grid. A logical constraint prevents the robots from occupying the same cell at any given time. Another rule prevents them from passing through each other, as this is not possible in the real world. Testing within this limited environment revealed critical and unexpected edge case issues: For example, making certain the robots do not wander off the grid, or attempting to pick up a shelf where there is none.

In order to expand automated warehouse activity, the third phase models a 4x4 grid, initializing the concept of a highway and specifying delivery locations called picking stations.

The fourth phase models the complete problem domain which includes orders that require fulfillment by product quantities on a plurality of shelves. The implemented encoding was tested with five predefined problem instances, which executed successfully. The results are itemized in Table 1.

Instance 1 tests for standard conditions to fulfill three orders with products situated on four of six shelves.
Instance 2 tests highway utilization and the fulfillment of two orders with products on three of five shelves.
Instance 3 requires the relocation of a shelf to retrieve another and tests fulfilling two orders delivered to the same picking station.
Instance 4 includes a condition where one of the shelves contains a product that is required by two orders at one picking station.
Instance 5 tests for conditions to fill one order at one picking station with items on two shelves.

Problem Instance	Makespan	Operations	CPU Time (s)
1	13	24	58.2
2	11	22	9.7
3	7	10	0.3
4	10	18	17.2
5	6	10	0.1

Table 1: Timing results with optimized makespan. In operations research, the makespan of a plan is the duration of time elapsed from start to end.

To investigate robot cooperation and highway utilization in repositioning shelves to retrieve one that is buried behind others, a sixth problem instance was developed. An early edition of the final encoding solved this test case with a makespan of 20 time steps and required 1,760.1 CPU seconds. The animated results captured from Asprilo can be viewed at: vimeo.com/516442641

Testing was performed on an 18-core iMac Pro with 128 GB of memory and on a 4-core Windows 7 personal computer with 12 GB of memory. CPU times are reported for the 4-core computer. Given that real-world warehouses would require a larger problem space, one concern the results seem to imply is the scalability factor for ASP solutions.

All resulting plans were verified using a solution checker from an earlier answer set programming competition[4]. Additionally, the plans were processed for import into Asprilo to visually inspect for anomalies.

Debugging is a trying process in the answer set programming domain. For example, problem instance 4 proved challenging for the preliminary set of order processing rules. This required

3 potassco.org/clingo/run

4 aspcomp2017.dibris.unige.it

several iterations to identify the reasons for vanishing answer sets. The action of removing products for two orders from a single shelf would not satisfy the rule body predicates.

To analyze the issue, code simplification was employed. Marginally similar to scoping a concern in procedural programming, here we must remove what is unnecessary to arrive at the fundamentals. For example, in the context of modeling action states, consider the following negative program which correctly produces four stable models, where *a* represents an action and *p* and *q* are states.

```
{a}.1{p0;q0}1.{q1}:-q0.{p1}:-p0.q1:-a.
:- not 1{p1;q1}1.
```

If we remove the *:- not* in the second line, the result is a positive program that produces six stable models, two of which are incorrect for action transitions. To investigate why this is, we can simplify the program to the following:

```
{a}. p:-a.
:- not p.
```

This is sated correctly by only one model *{p, a}*. If we again remove the *:- not*, then two models *{p}* and *{p, a}* satisfy the resulting program. In the context given, set *{p}* is not allowable. The difference is that the positive program imposes an assertion whereas the negative program delineates possibilities through logically sound inference.

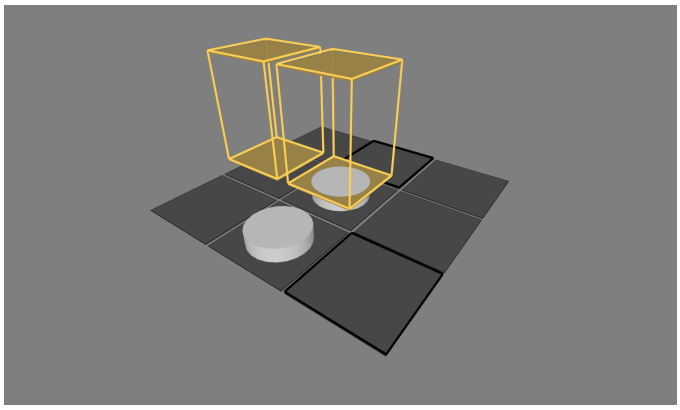

Figure 3: WebGL visualization of a minimal instance with p5js.

IV. Lessons Learned

Developing code is complex and challenging, yet rewarding when it works as planned. Learning a new coding language and its underlying foundation mimics real world needs. Perhaps an obtuse language at inception, clingo enables the expression of a system of assertions and logical inferences to model complicated and dynamic physical structures.

This study presents an approach to solving a warehouse automation problem using answer set programming for optimal pathfinding. A key takeaway is the practice of code analysis through simplification. Further, utilizing a visualization tool such as Asprilo is a critical method to view the output of the code to verify its proper action and to identify faults in operation. Unification of individual components is integral to enterprise activity, especially in the context of warehouse retrieval.

This project has provided a deeper understanding of how logically consistent unit operations [6] can structurally form a more reliable basis for a stable system operation as a whole.

The work described in this paper can be applied to a myriad of autonomous systems. For instance, within a research project at kynamatrix[5], we are investigating personal autonomous transportation methods in the context of a rapidly deployable emergency urban environment. The closed arrangement is well-defined, akin to the controlled world of a smart warehouse or factory. Although larger in scale, the same principles apply, in that answer set programming can potentially provide additional solution options toward planning the movement of robotic vehicles within a small city. A valuable facet of this research is the intersection of ASP and the robot operating system[6] framework which has been investigated by Andres et al [7]. Additionally, two types of three-dimensional aspects can be explored. First, extending visualization of the warehouse using tools such as threejs[7] or p5js[8] to render a three-dimensional representation as depicted in Fig. 3. Second, to the degree that warehouses optimize their space, robots will travel in the third dimension to reach shelves at higher levels or on different floors.

As automated scalable warehouses become collaborative systems of robotic manipulators, knowledge representation and reasoning tools can assist in planning the optimal movement of products from point A to point B.

V. References

[1] Lifschitz, V. 2019. "Dynamic Systems." In *Answer Set Programming*, pp. 129-146. Springer, Cham. doi.org/10.1007/978-3-030-24658-7_8

[2] Erdem, E.; Kisa, D.; Oztok, U.; and Schüller, P. 2013. A general formal framework for pathfinding problems with multiple agents. In *Proceedings of the Twenty-Seventh National Conference on Artificial Intelligence* (AAAI'13), M. desJardins and M. Littman, Eds. AAAI Press, 290-296.

[3] Schäpers, B.; Niemueller, T.; Lakemeyer, G.; Gebser, M.; and Schaub, T. 2018. ASP-based time-bounded planning for logistics robots. In *Proceedings of the International Conference on Automated Planning and Scheduling* (Vol. 28, No. 1).

[4] Gebser, M.; Obermeier, P.; Otto, T.; Schaub, T.; Sabuncu, O.; Nguyen, V.; and Son, T.C. 2018. Experimenting with robotic intra-logistics domains. *Theory and Practice of Logic Programming*. 18. 502-519. doi.org/10.1017/S1471068418000200.

[5] Gelfond, M. and Watson, R., 1999. On Methodology of Representing Knowledge in Dynamic Domains. *Electronic Notes in Theoretical Computer Science*, 25, pp.121-132.

[6] Bogost, I. 2008. *Unit Operations: An approach to videogame criticism.* MIT Press.

[7] Andres, B.; Rajaratnam, D.; Sabuncu, O.; and Schaub, T. 2015. September. Integrating ASP into ROS for reasoning in robots. In *International Conference on Logic Programming and Nonmonotonic Reasoning* (pp. 69-82). Springer, Cham. doi.org/10.1007/978-3-319-23264-5_7

5 kynamatrix.org
6 ros.org
7 threejs.org
8 p5js.org