UNIVERSITY OF AMSTERDAM

Knowledge Representation

QR Project Report

Andreas Hadjipieri
Tomáš Fábry

October 27, 2017

1 Assumptions

For this project we have decided to use the recommended assumptions given by the course document and included some of our own. The specifics are as follows:

- Quantities
  - Inflow
  - Volume
  - Outflow
  - Height
  - Pressure
- Quantity spaces
  - Inflow: [0, +]
  - Volume: [0, +, max]
  - Outflow: [0, +, max]
  - Height: [0, +, max]
  - Pressure: [0, +, max]
- Dependencies
  - I+(Inflow, Volume)
  - I-(Outflow, Volume)
  - P+(Volume, Outflow)
  - VC(Volume(max), Outflow(max))
  - VC(Volume(0), Outflow(0))
  - P+(Volume, Pressure)
  - P+(Volume, Height)
  - VC(Volume(max), Pressure(max))
  - VC(Volume(0), Pressure(0))
  - VC(Volume(max), Height(max))
  - VC(Volume(0), Height(0))

Aside of the listed formal assumptions, we assume that when a quantity reaches its maximum magnitude, the derivative automatically changes from + to 0 and similarly, when the magnitude drops to 0, the negative derivative changes to 0 at the same time. This way, we don't explicitly model real life phenomena such as overflow of water out of the container, we simply state that the volume and outflow are not increasing anymore when they reach maximum, but inflow can indeed grow greater than that.

Furthermore, one of the assumptions that is inherent to qualitative reasoning but is not obvious is that when there is a quantity with a point value as magnitude and a non-zero derivative, the next state has to reflect the change of the magnitude. A simple example being the transition from State 1 to State 2 in our submitted state graph. Because the magnitude of Inflow is 0 and its derivative is +, the next state needs to have a positive magnitude. During this transition there is indeed some time that passes in real life, but the duration is not long enough for the model to allow a change in the exogenous variable. In other words, during such transition, it is not allowed to change the exogenous variable(with the only exceptions being propagating dependencies if necessary).

Inbetween each state transition, all dependencies are propagated to change the respective quantities. We assume that some time has passed during each transition, but the amount of time passed is not explicitly stated therefore multiple ways to resolve derivatives to magnitudes is possible. This is described in more detail in the following chapters.

2 The Algorithm

The algorithm to create the state graph is a depth-first tree search through states, that are being dynamically generated from one to another given the dependencies and the current state, starting with an initial state of every quantity being (0, 0).

The graph is an adjacency matrix data structure, where for each unique state we keep a list of states that we can get to, given the rules of the model. First, an initial state is used to generate all possible states that can be created from it. This is done by first resolving derivatives to magnitudes in all combinations that are possible. If the magnitude is a range value and the derivative is non-zero, the magnitude can either remain the same range value or change to any adjacent point values. Then for all combinations of magnitudes, derivatives that satisfy the rules are generated. If there was a case where a point value magnitude had a non-zero derivative, no change is allowed on the exogenous variables, as stated in chapter 1. With the exception being when the magnitude reaches max or 0.

The combinations of magnitudes and derivatives for each quantity are put into states and fed through all dependencies that check if such state is allowed. This is our way of resolving all the dependencies at once, not one by one - the states that would not be possible given the dependencies are discarded. For example, when the magnitude of Inflow is 1 and the magnitude of Outflow is 0, then the derivative of Volume must be + because of I+(Inflow, Volume) and I-(Outflow, Volume). Unless the magnitude is Max, which automaticallychanges the derivative from + to 0.

The newly generated states are added as neighbours of the current state and if they have not been already expanded(generated neighbours), they are also added to the first dimension of the graph. Then the algorithm takes the next state along the 1st dimensions to generate its neighbours and so on until there are no more states to expand. In the end, we output the graph as an image and the inference trace as a text file, where every state transition is explained in terms of quantities and their dependencies.

Algorithm 1 Create Graph

```
1: Initialize initialState, list of dependencies, empty graph
2: while There is a state that has not been expanded yet do
3:     Generate valid magnitudes for every quantity
4:     Generate valid derivatives for every quantity
5:     Create list of new states using all permutations of the new quantities
6:     for Every state in the list do
7:         for Every VC dependency do
8:             if State doesn't satisfy the dependencies then
9:                 discard state
10:
11:         for Every Proportional dependency do
12:             Group together dependencies with the same target quantity
13:             if State doesn't satisfy the dependency group then
14:                 discard state
15:
16:         for Every Influence dependency do
17:             Group together dependencies with the same target quantity
18:             if State doesn't satisfy the dependency group then
19:                 discard state
20:
21:         for Every Quantity in parentState do
22:             if (Magnitude was 0 and derivative + or magnitude was Max and
    derivative -) then
23:                 if Action was taken on the exogenous variable(Inflow) then
24:                     discard state
25:
26:     for Every new valid state do
27:         if state hasn't been used to generate neighbours yet then
28:             Add state to the first dimension of the graph
29: return graph
```

3 Implementation

We used Python as the primary programming language, with no special frameworks or libraries except graphviz, which was used to create a visual representation of the state graph. In order to run the program, it is necessary to install graphviz on the operating system(and add it to the PATH variable on windows) and install all the imported packages using a python package manager of choice.

The code was initially built to be a general-purpose state machine, however, as time progressed towards the deadline, the function propagateI() to propagate "Influence" dependencies had to be fitted specifically to our problem of sink modeling. The rest of the functions are general enough that adding additional quantities, dependencies or changing the initial state should cause few to zero problems, although this functionality has not been thoroughly tested, aside from adding Pressure and Height with the respective dependencies.

In the code we use 3 classes and multiple files with functions to make the code more readable:

- Class Quantity
  For individual quantities in a state, each having a name, value, derivative, range and a boolean value to indicate whether it is exogenous or not.

- Class State
  In class State we most importantly keep a list of quantities. The number is therefore dynamic and everywhere else in the code, we loop through all quantities and all relationships in order to generate neighbouring states. State also keeps a list 'reasons', which contains the explanation of every quantity - how it got generated from its parent state.

- Class Relationship
  List of all instances of class Relationships is kept throughout the whole program, with a source and a target quantity, indicator for the type of quantity and its sign(+ or -).

We found it redundant to include a class diagram because the classes are very independent. Main.py serves the purpose of initializing the initial state, quantities, dependencies and calling the generating and printing functions. Generator.py is the core of the algorithm, where function createGraph() calls generateStates() in a loop until there are no more states to expand, while also checking for uniqueness and assigning id's to states to be later used in plotting. The generateStates() function resolves derivatives to magnitudes as described in the previous chapter and calls functions to propagate every type of dependency in the list of dependencies. These functions remove invalid states from the list of newly generated states and finally a list of valid neighbouring states is returned and added to the graph. Lastly, functions from Plot.py are used to plot the graph as an image using graphviz.

4 Outputs

The graph is too large for a single A4 page therefore it is recommended to view it directly with a proper image viewing software to be able to zoom in on nodes and edges. The direction of edges is indicated with arrows. There is no specific end state because the exogenous variable can force a transition from most states. We have 20 states total. We have also included the graph in text form which might be more readable for some people given the size of the graph.

Example: $0 \rightarrow [0, 1]$ indicates that from State 0 we can stay at State 0 or go to State 1. After this structure, each state is printed with its quantities.

The inference trace in text files shows all generated, neighbouring, states for every unique state and gives explanation for how every quantity changed and why. Because of similar behavior of Pressure and Height quantities, the explanations might sometimes be redundant, but we have kept every information for the sake of completeness. To structure the text file, we denote the beginning of a new unique state with * symbols and the beginning of a new explanation for every unique state with - symbols.

5 Conclusion

In this project report we have successfully generated a state graph as well as an inference trace in text form, verified and revised it multiple times with respect to our assumptions and found both to be correct. We have worked with and learned the core QR principles, implemented the given extra details as well as described the flow of our program.