A Qualitative Reasoning Model of a simple (?) Container System

Project Report

Jonsson, Haukur	Rajamanickam, Santhosh	Rapp, Max
11137304	11650702	11404310

October 27, 2017

1 Introduction

Everyday life requires engaging with countless systems of varying complexity. Successful engagement necessitates robust approximate prediction of their behaviour. However, understanding the dynamics of even the simplest of these systems would require solving a set of differential equations of prohibitive complexity.

Rather than precise modelling, humans therefore engage in *Qualitative Reasoning* to understand and predict the behaviour of systems surrounding them. Qualitative models are built by using simplifying *assumptions* to construct a simplified ontology of the system including its objects and the causal relations between them as well as discretizing the quantities of the objects and the magnitudes of the relations. Such models are surprisingly powerful in predicting possible system outcomes.

It is threfore arguable that understanding a domain is a function of one's qualitative model of that domain. In this paradigm, human (and machine) learning and teaching can then be understood in terms of building and adapting qualitative models of the world.

The computer assisted learning paradigm tries to leverage this approach by developing *Human Level AI* that assists and interacts with the learner in building better qualitative models.

The goal of this project is to implement a qualitative model of a container system that could serve in an interactive learning software by taking various inputs and returning a state graph and trace of the development of the system.

2 Two Container Models

The system to be modelled can be described as follows: A **tap** is used to regulate the influx of water into an open **container** with a **drain** through which water escapes.

The basic ingredients needed to build a model for this system are the *entities* which describe the basic objects the model contains. An entity may have one or more *quantities* describing its magnitude. The quantities each have a magnitude constrained by a *quantity space* containing its allowed values as well as a *derivative* with quantity space (-,0,+) describing the current change in quantity. Between quantities *causal relations* obtain. In our case these include *influences* (I^+, I^-) and *proportionalities* (P^+, P^-). Influences are positive or negative relationships from the magnitude of a quantity to the derivative of another. Proportionalities are positive or negative relationship from one derivative to another. Another component we need are *correspondences*. Correspondences can be implications or equivalences between values of quantities' magnitudes.

Using these ingredients, depending on one's assumptions, models of varying complexity of the container system can be devised. Here we describe two.

2.1 Model I

Ontology:

- Entities: Tap, Container, Drain
- Quantities(Spaces): Inflow (Zero; Plus), Volume (Zero, Plus, Max), Outflow (Zero, Plus, Max)
- Relations: I^+(Inflow, Volume), I^-(Outflow, Volume), P^+(Volume, Outflow)
- Correspondences: Volume(Zero)$\leftrightarrow$Outflow(Zero), Volume(Max)$\leftrightarrow$Outflow(Max)

The ontology above is depicted in figure 1. To study the development of the system we not only need an ontology but also an *initial state* (or *scenario*). figure 2 represents the scenario which we aim to implement for Model I. Namely, we assume that inflow and outflow are at zero in the beginning and that the inflow is exogenuously determined by a function taking the form of a positive parabola. I.e. δInflow will initially increase but eventually stagnate and start to decrease until it reaches zero.

2.2 Model II

Ontology:

- Entities: Tap, Container, Drain
- Quantities(Spaces): Inflow (Zero; Plus), Height(Zero, Plus, Max), Pressure (Zero, Plus, Max), Volume (Zero, Plus, Max), Outflow (Zero, Plus, Max)
- Relations: I^+(Inflow, Volume), I^-(Outflow, Volume), P^+(Volume, Outflow)

The more complex ontology of Model II is shown in figure 3 and the corresponding initial state (unchanged except for entities) in figure 4.

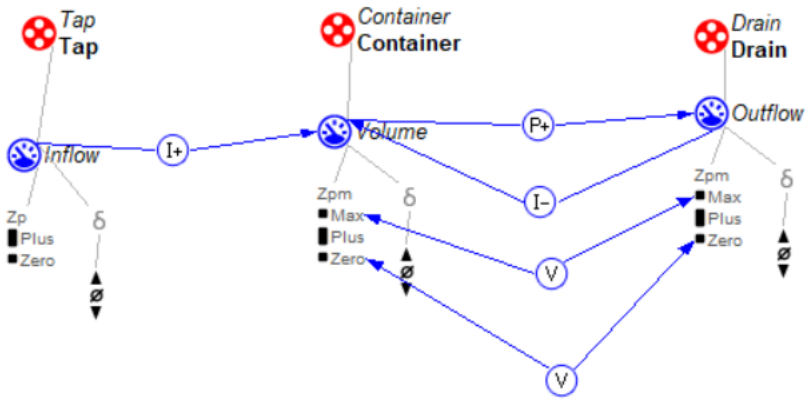

Figure 1: Ontology of Model I. Note the red entities which indicate conditions that have to be met for the relations to apply

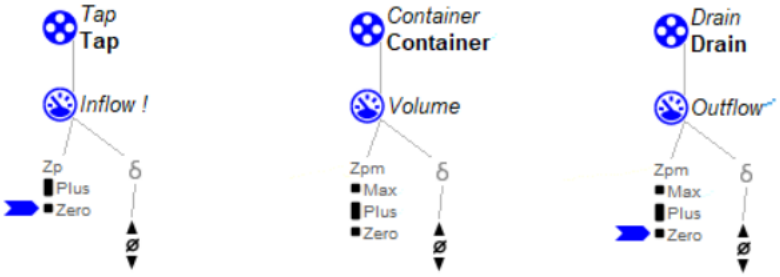

Figure 2: Initial state of Model I

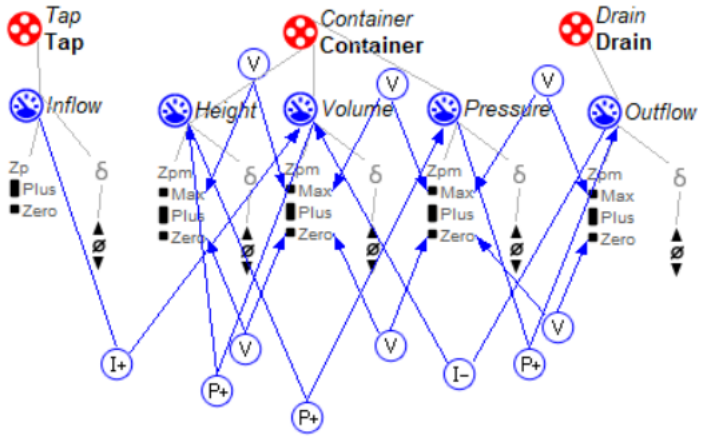

Figure 3: In the more complex Model II, outflow is no longer determined by volume but by pressure which is in turn determined by the height of the fluid in the container which is proportiional to the volume.

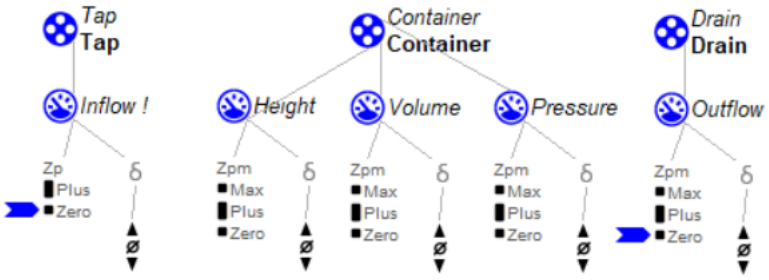

Figure 4: Initial state of Model II

3 Target Simulation and State Graph

to begin with, we set up Model I and Model II in the QR-software *Garp3* to create benchmarks for our implementations. Simulations using default modelling settings (figure 5) lead in both cases to state graphs containing 13 nodes. As expected, the unique end state (state 12 in both cases) consists of an empty container with zero in- and outflow (figure 6). Interestingly, the graphs of the two models look completely identical inspite of the added quantities in Model II (figure 7).

4 Implementation

4.1 Modelling Assumptions

We now set out to recreate the state graphs of Model I and II using a much simpler implementation. Namely, of the simulation preferences described in figure 5 we will only implement the following:

Basic Epsilon Ordering: Any termination that consists of a point-value changing to an interval is immediate and takes priority over changes that take time such as interval to landmark changes. if more that one immediate terminations is applicable, ... In this way we divide the termination calculation into two phases *static changes* and *dynamic changes*. Branching can take place only in the latter phase.

Derivative values are point values: or equivalently proportionalities act immediately. This assumption can be justified by the fact that proportionalities are usually the result of having different descriptions of the same state of a system that are pragmatically useful but ontologically reducible to each other. In our case, consider the water column in the container in Model II: the height of the column, its volume and the pressure in the column are three different yet dependent ways of describing the same state of the system; thus if one of these quantities changes, so do the others by definition. Proportionality here thus merely describes the same change by different names and therefore acts immediately.

After inferring terminations, in a third phase, we check the consistency of the termination candidates against the following conditions:

No magnitude change without non-zero derivative: a magnitude of a quantity can only increase/decrease if its derivative is greater/smaller than zero.

No derivative change without (exogenous) influence: likewise, we don't allow derivative changes that are not caused by an influence, proportionality or exogenous variable.

The two preceding conditions immediately imply the next one:

No causation by correspondence: implications and equivalences act as sanity checks and let us throw out inconsistent states but cannot be used in the static/dynamic phases.

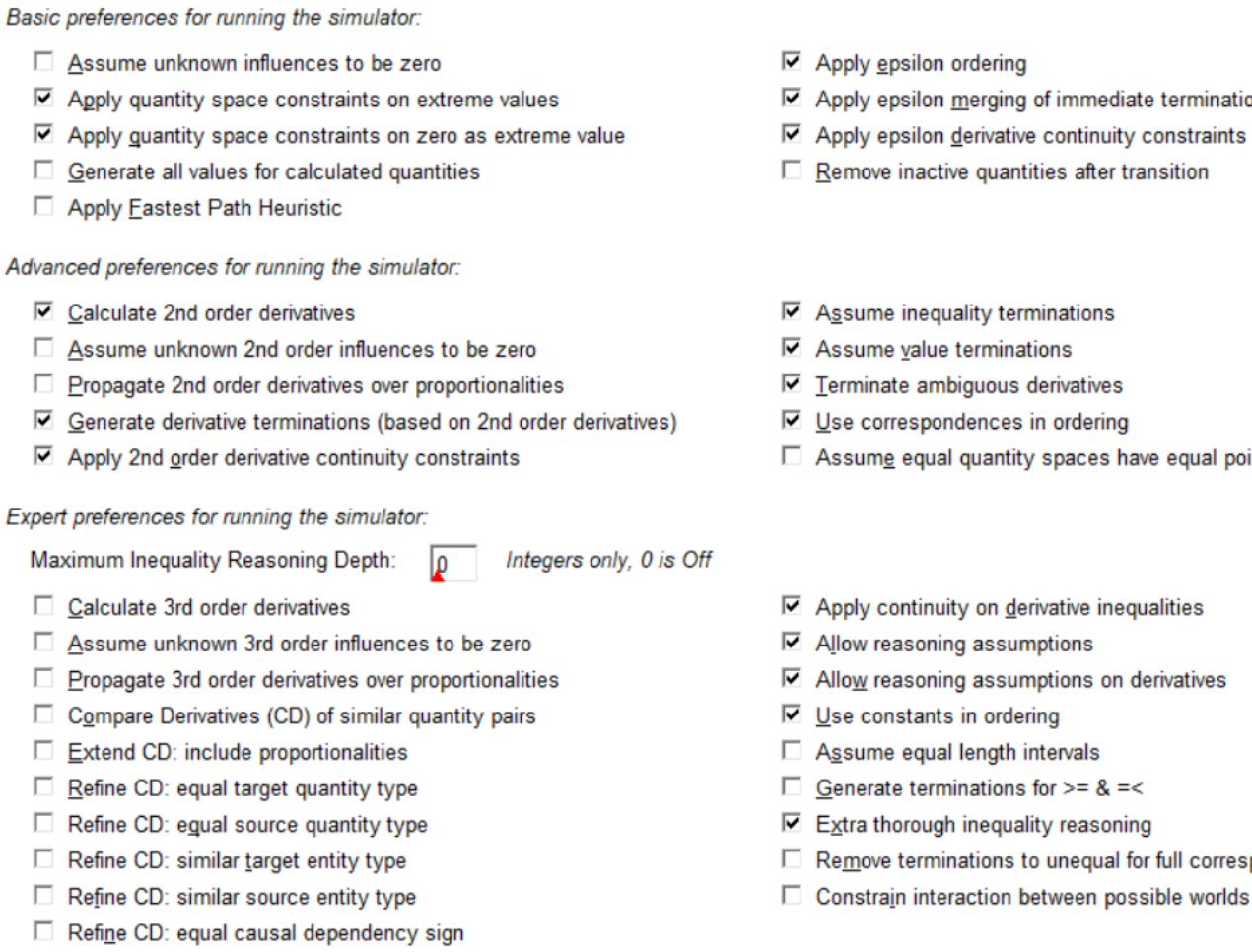

Figure 5: The modelling preferences used in our benchmark simulations.

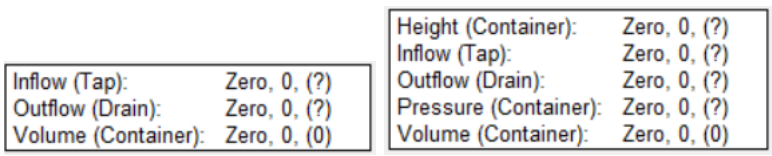

Figure 6: The end states of the simulations of Model I and II respectively.

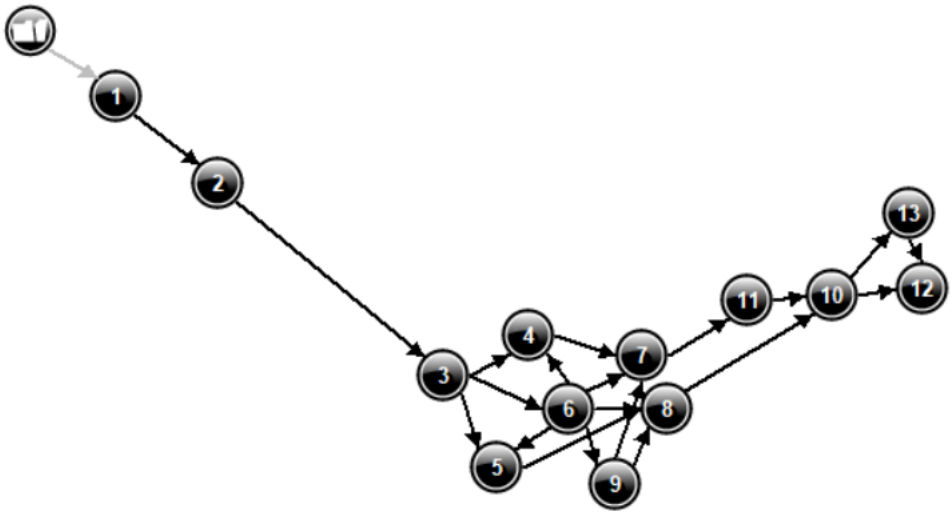

Figure 7: State graph for Model I and II. While the states represented differ in the number of quantities they represent, the identity of the graph representations indicate that Model I and Model II describe the same system in two different yet - with respect to evolution - equivalent ways.

No fleeting equalities: If $I^+(A,B)=I^-(C,B)$ and $\delta A \neq \delta C$, $\delta B \neq 0$. The justification for this assumption is that such points of fleeting inequality should take priority by epsilon ordering since moving away from a point of equality is immediate. Therefore they should immediately terminate into a different state. However, they are not spotted by our basic epsilon ordering and hence we discard them here.

4.2 Algorithm

Note that these assumptions do not touch on issues such as second order derivatives or inequality reasoning. Our algorithm is thus much more primitive than Garp3's. We describe how it works by the following pseudocode:

enter pseudocode here

5 Results

Which deviations from the benchmark simulations will our modelling choices incur? And do they affect the overall degree to which our algortihm captures the behaviour of the system?

The results of our simulations are visualized by the state graph in figure 8 as well as the state descriptions in table 1 and 2. Consistently with the Garp3 simulations we found that the state graphs of Model I and Model II coincide.

Algorithm 1 Construct State Graph

```
1: procedure MakeGraph(intialState,causalGraph)
2:     stateGraph ← graph.addHead(initialState)
3:     stateStack ← stack.push(initialState)
4:     while stack.length(stateStack)! = 0 do
5:         currentState ← stack.pop(stateStack)
6:         nextStates[] ← computeNextStates(currentState, causalGraph)
7:         for index ← 0 to length(nextStates) do
8:             childState ← nextStates[index]
9:             exists ← checkStateExists(state)
10:            if notexists then
11:                assignStateNumber(childState)
12:                graph.addChild(stateGraph,currentState,childState)
13:                stack.push(stateStack,childState)
14:            else
15:                graph.addChild(stateGraph,currentState,childState)
16:    return stateGraph
```

Algorithm 2 Compute next state transitions

```
1: procedure computeNextStates(currentState,causalGraph)
2:     newState ← NULL
3:     newState ← applyPointChanges(currentState, causalGraph)
4:     if newState not NULL then
5:         newState ← applyStaticChanges(newState, causalGraph)
6:         return newState
7:     newStatesList[ ]
8:     counter ← 0
9:     newStatesList ← applyIntervalChanges(currentState, causalGraph)
10:    for index ← 0 to length(nextStatesList) do
11:        newState ← applyStaticChanges(newStatesList[i], causalGraph)
12:        isConsistent ← checkStateConsistency(newState)
13:        if isConsistent then
14:            newStatesList[counter] ← newState
15:            counter++
16:    return newStatesList
```

Algorithm 3 Applying Point Changes

```
1: procedure applyPointChanges(currentState,causalGraph)
2:     newState ← currentState
3:     for index1 ← 0 to length(causalGraph.entities) do
4:         for index2 ← 0 to length(causalGraph.entities[index1].quantaties)
   do
5:             if newState[index1][index2].derivavtive == 0 then
6:                 if newState[index1][index2].magnitude == 0 then
7:                     applyDerivative(newState[index1][index2])
8:                 if currentState[index1][index2].magnitude == MAX then
9:                     applyDerivative(newState[index1][index2])
10:    for index1 ← 0 to length(causalGraph.relationships) do
11:        if causalGraph.relationships[index1] == "Influence" then
12:            index2, index3 ← causalGraph.relationship[index1].recievingPartyIndices
13:            if currentState[index2][index3].derivative == 0 then
14:                applyInfluenceRelationship(newState)
15:    return newState
```

Algorithm 4 Applying Static Changes

```
1: procedure applyStaticChanges(currentState,causalGraph)
2:     newState ← currentState
3:     condtion ← TRUE
4:     while condtion == TRUE do
5:         for index1 ← 0 to length(causalGraph.relationships) do
6:             beforeState ← newState
7:             if causalGraph.relationships[index1] == "Proportional" then
8:                 applyProportionalRelationship(newState)
9:             if causalGraph.relationships[index1] == "Equivalence" then
10:                applyEquivalenceRelationship(newState)
11:            condition ← ifDifferent(beforeState, newState)
12:    return newState
```

Algorithm 5 Applying Interval Changes

```
1: procedure applyStaticChanges(currentState,causalGraph)
2:     newState ← currentState
3:     newStatesList[ ]
4:     counter = 0
5:     for index1 ← 0 to length(causalGraph.relationships) do
6:         if causalGraph.relationships[index1] == "Influence" then
7:             applyInfluenceRelationship(newState)
8:         if ifDifferent(nextState, currentState) then
9:             newStatesList[counter] ← newState
10:            counter + +
11:    newState ← currentState
12:    for index1 ← 0 to length(causalGraph.entities) do
13:        for index2 ← 0 to length(causalGraph.entities[index1].quantaties)
   do
14:            applyDerivative(newState[index1][index2])
15:            if ifDifferent(nextState, currentState) then
16:                newStatesList[counter] ← newState
17:                counter + +
18:    newState ← currentState
19:    newState ← applyExogenous(causalgraph.typeOfExogenous, newState)
20:    if ifDifferent(nextState, currentState) then
21:        newStatesList[counter] ← newState
22:        counter + +
23:    return newStatesList
```

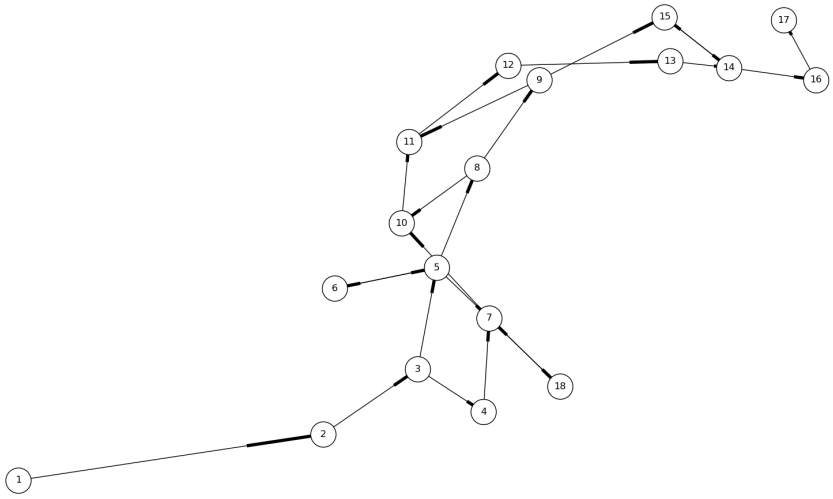

Figure 8: State graph for Model I and II according to our simulation.

Table 1: State table of Model I

#	Inflow	δ Inflow	Volume	δ Volume	Outflow	δ Outflow	Children
1	0	+	0	0	0	0	2
2	+	+	0	+	0	+	3
3	+	+	+	+	+	+	4, 5
4	+	+	Max	+	Max	+	7
5	+	0	+	+	+	+	6, 7, 8
6	+	0	+	0	+	0	5
7	+	0	Max	+	Max	+	10, 18
8	+	-	+	+	+	+	9, 10
9	0	-	+	+	+	+	11, 15
10	+	-	Max	+	Max	+	11
11	0	-	Max	+	Max	+	12
12	0	-	Max	0	Max	0	13
13	0	-	Max	-	Max	-	14
14	0	-	+	-	+	-	15, 16
15	0	-	+	0	+	0	14
16	0	-	0	-	0	-	17
17	0	-	0	0	0	0	End
18	+	0	Max	0	Max	0	7

Table 2: State table of Model II

#	Inflow	δ Inflow	Height	δ Height	Pressure	δ Pressure	Volume	δ Volume	Outflow	δ Out
1	0	+					0	0	0	0
2	+	+					0	+	0	+
3	+	+					+	+	+	+
4	+	+					Max	+	Max	+
5	+	0					+	+	+	+
6	+	0					+	0	+	0
7	+	0					Max	+	Max	+
8	+	-					+	+	+	+
9	0	-					+	+	+	+
10	+	-					Max	+	Max	+
11	0	-					Max	+	Max	+
12	0	-					Max	0	Max	0
13	0	-					Max	-	Max	-
14	0	-					+	-	+	-
15	0	-					+	0	+	0
16	0	-					0	-	0	-
17	0	-					0	0	0	0
18	+	0					Max	0	Max	0