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

Project Report

Jonsson, Haukur
11137304

Rajamanickam, Santhosh
11650702

Rapp, Max
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 therefore 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

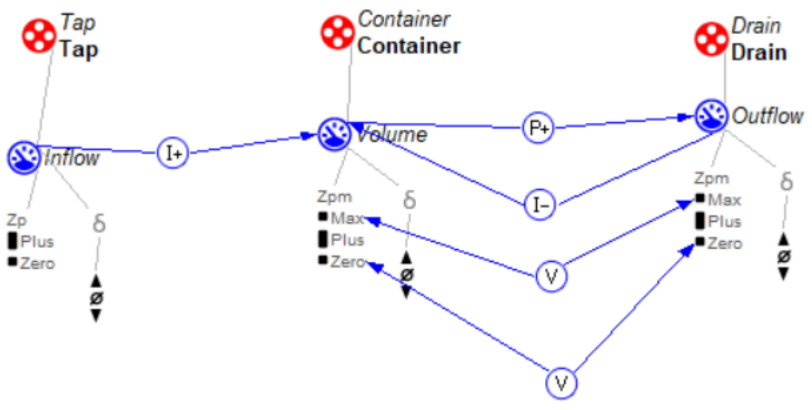


Figure 1: Ontology of Model I. Note the red entities which indicate conditions that have to be met for the relations to apply.

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)↔Outflow(Zero), Volume(Max)↔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 exogenously 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.

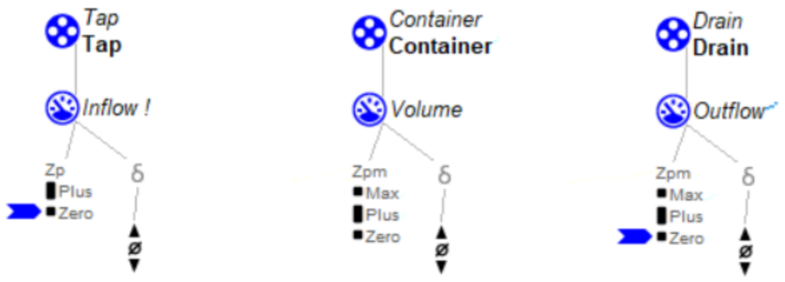


Figure 2: Initial state of Model I.

2.2 Model II

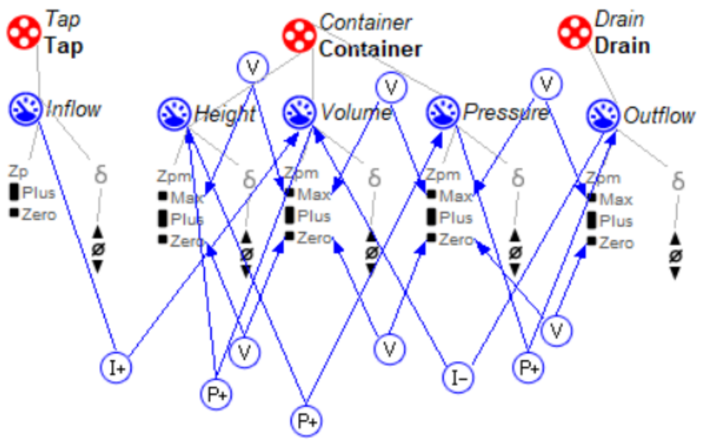


Figure 3: In the more complex Model II, outflow is no longer determined by volume but by pressure which is in turn determined by the height of the fluid in the container which is proportional to the volume.

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.

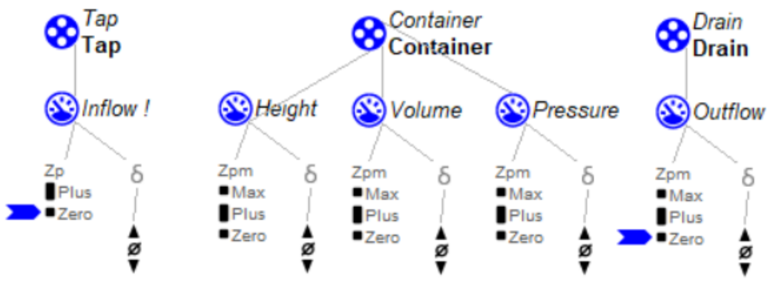


Figure 4: Initial state of Model II.

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 7). Interestingly, the graphs of the two models look completely identical inspite of the added quantities in Model II (figure 6).

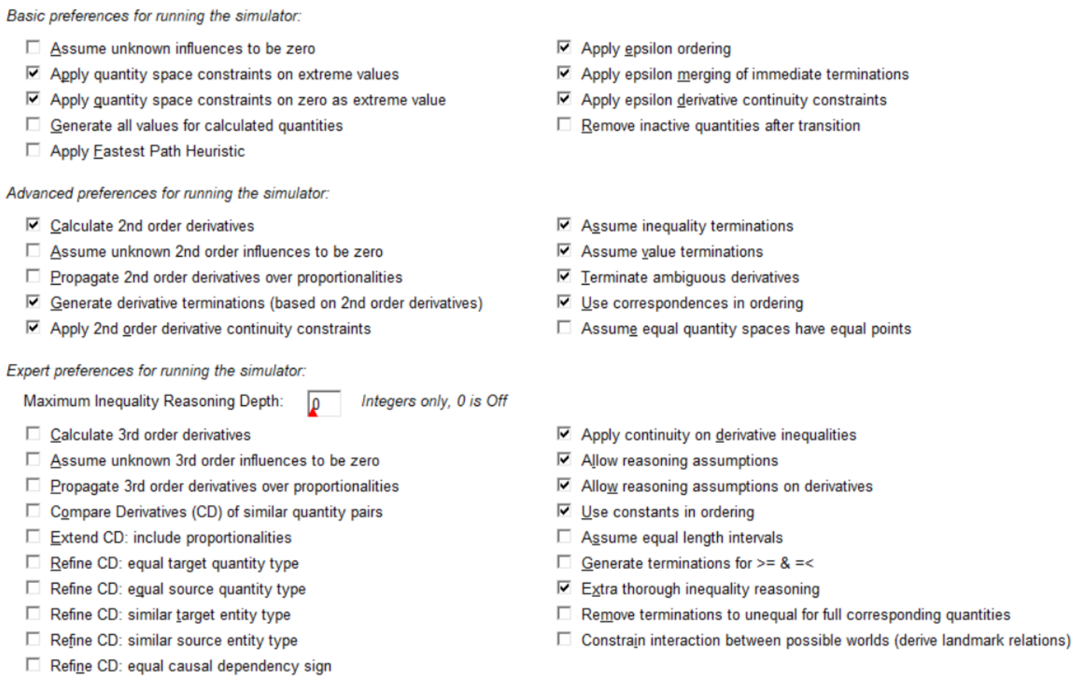
4 Implementation

4.1 Environment

To implement a QR-reasoner for Model I and II we used Python 3.6 including the packages *networkx*, *numpy* and *matplotlib*. To run it use the command line command "python run_model.py". Note that uncommenting the graph drawing function may lead to dependency issues.

4.2 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



Basic preferences for running the simulator:

☐ Assume unknown influences to be zero	☑ Apply epsilon ordering
☑ Apply quantity space constraints on extreme values	☑ Apply epsilon merging of immediate terminations
☑ Apply quantity space constraints on zero as extreme value	☑ Apply epsilon derivative continuity constraints
☐ Generate all values for calculated quantities	☐ Remove inactive quantities after transition
☐ Apply Fastest Path Heuristic	

Advanced preferences for running the simulator:

☑ Calculate 2nd order derivatives	☑ Assume inequality terminations
☐ Assume unknown 2nd order influences to be zero	☑ Assume value terminations
☐ Propagate 2nd order derivatives over proportionalities	☑ Terminate ambiguous derivatives
☑ Generate derivative terminations (based on 2nd order derivatives)	☑ Use correspondences in ordering
☑ Apply 2nd order derivative continuity constraints	☐ Assume equal quantity spaces have equal points

Expert preferences for running the simulator:

Maximum Inequality Reasoning Depth: 0 *Integers only, 0 is Off*

☐ Calculate 3rd order derivatives	☑ Apply continuity on derivative inequalities
☐ Assume unknown 3rd order influences to be zero	☑ Allow reasoning assumptions
☐ Propagate 3rd order derivatives over proportionalities	☑ Allow reasoning assumptions on derivatives
☐ Compare Derivatives (CD) of similar quantity pairs	☑ Use constants in ordering
☐ Extend CD: include proportionalities	☐ Assume equal length intervals
☐ Refine CD: equal target quantity type	☐ Generate terminations for >= & =<
☐ Refine CD: equal source quantity type	☑ Extra thorough inequality reasoning
☐ Refine CD: similar target entity type	☐ Remove terminations to unequal for full corresponding quantities
☐ Refine CD: similar source entity type	☐ Constrain interaction between possible worlds (derive landmark relations)
☐ Refine CD: equal causal dependency sign	

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

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 than one immediate terminations is applicable we apply them in the ordering specified in Algorithm 3. In this way we divide the termination calculation into two phases *intra-state changes* and *inter-state changes*. Branching can take place only in the latter phase.

Derivative values are point values: Note that this entails that 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:

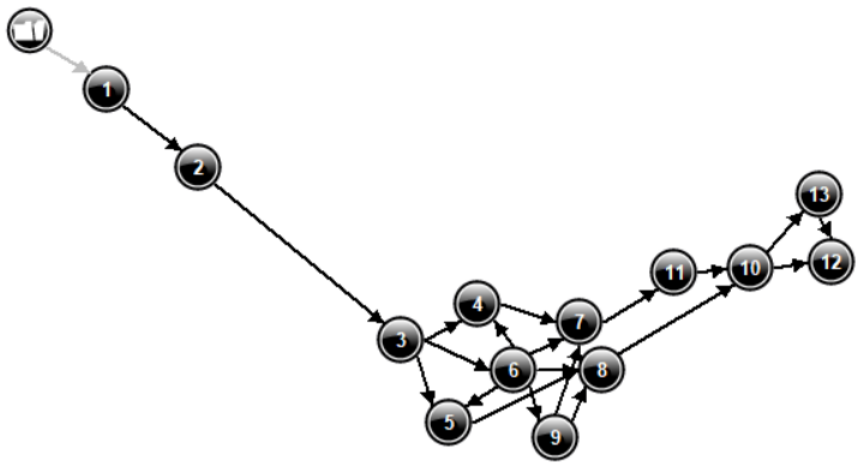


Figure 6: State graph for Model I and II. While the states represented differ in the number of quantities they represent, the identity of the graph representations indicate that Model I and Model II describe the same system in two different yet - with respect to evolution - equivalent ways.

No 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.

No fleeting equalities: If $I^{+}(A, B)$, $I^{-}(C, B)$, $A = C$ 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.

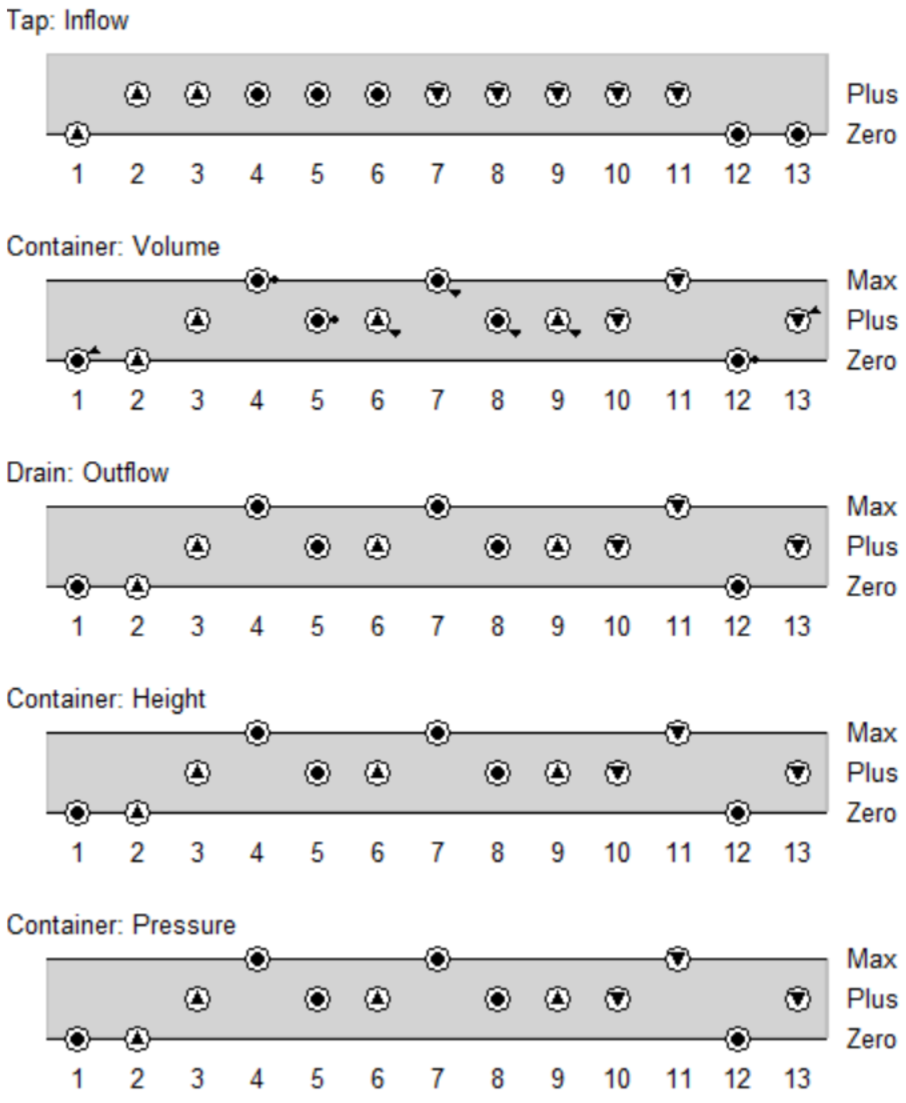


Figure 7: Value history of Model I & II (lower part). Notice that state 12 is the end state in both cases.

4.3 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:

Algorithm 1 Construct State Graph

```
procedure MakeGraph(intialState,causalGraph)
    $stateGraph \leftarrow$ graph.addHead($initialState$)
    $stateStack \leftarrow$ stack.push($initialState$)
    while stack.length(stateStack)! = 0 do
        $currentState \leftarrow$ stack.pop($stateStack$)
        $nextStates[] \leftarrow$ computeNextStates($currentState, causalGraph$)
        for $index \leftarrow 0$ to length($nextStates$) do
            $childState \leftarrow nextStates[index]$
            $exists \leftarrow$ checkStateExists($state$)
            if not$exists$ then
                assignStateNumber($childState$)
                graph.addChild($stateGraph, currentState, childState$)
                stack.push($stateStack, childState$)
            else
                graph.addChild($stateGraph, currentState, childState$)
    return $stateGraph$
```

Algorithm 1: makeGraph(initialState,causalGraph)

- This is the method that is used to create the state graph.
- It takes the *initialState* and *causalGraph* as parameters.
- The *causalGraph* contains information about the entities, their quantities and the corresponding relationships between them.
- The algorithm loops until we get all possible states by computing the possible states that each state can transition to and add them as children to the corresponding state.
- The algorithm also checks whether a state has already been generated previously and if it is already present we just add it as the child to the state from which it transitions from.

Algorithm 2: computeNextState(currentState,causalGraph)

- This is the method that is used to generate the possible state transitions from a single node.
- There are three possible updates that can be applied to a state which are: static changes which correspond to intra-state changes as well as point changes and interval changes which correspond to inter-state changes.
- After we get the possible transition states we check whether the states are consistent or not using the function *checkStateConsistency*

Algorithm 3: applyPointChanges(currentState,causalGraph)

Algorithm 2 Compute next state transitions

```
procedure COMPUTENEXTSTATES(currentState,causalGraph)
    $newState \leftarrow$ NULL
    $newState \leftarrow$ applyPointChanges($currentState, causalGraph$)
    if newState not NULL then
        $newState \leftarrow$ applyStaticChanges($newState, causalGraph$)
        return $newState$
    $newStatesList$[ ]
    $counter \leftarrow 0$
    $newStatesList \leftarrow$ applyIntervalChanges($currentState, causalGraph$)
    for $index \leftarrow 0$ to length($nextStatesList$) do
        $newState \leftarrow$ applyStaticChanges($newStatesList[i], causalGraph$)
        $isConsistent \leftarrow$ checkStateConsistency($newState$)
        if $isConsistent$ then
            $newStatesList[counter] \leftarrow newState$
            counter++
    return $newStatesList$
```

Algorithm 3 Applying Point Changes

```
procedure APPLYPOINTCHANGES(currentState,causalGraph)
    $newState \leftarrow currentState$
    for $index1 \leftarrow 0$ to length($causalGraph.entities$) do
        for $index2 \leftarrow 0$ to length($causalGraph.entities[index1].quantaties$) do
            if $newState[index1][index2].derivavtive == 0$ then
                if $newState[index1][index2].magnitude == 0$ then
                    applyDerivative($newState[index1][index2]$)
                if $currentState[index1][index2].magnitude == MAX$ then
                    applyDerivative($newState[index1][index2]$)
    for $index1 \leftarrow 0$ to length($causalGraph.relationships$) do
        if $causalGraph.relationships[index1] ==$ "$Influence$" then
            $index2, index3 \leftarrow causalGraph.relationship[index1].recievingPartyIndices$
            if $currentState[index2][index3].derivative == 0$ then
                applyInfluenceRelationship(newState)
    return $newState$
```

- This method applies magnitude changes of the quantities corresponding to its derivatives.
- This method also applies potential influence changes by comparing the derivative of the receiving quantity and the magnitude of the causal quantity.

Algorithm 4: applyStaticChanges(currentState,causalGraph)

Algorithm 4 Applying Static Changes

```
procedure applyStaticChanges(currentState,causalGraph)
    $newState \leftarrow currentState$
    $condtion \leftarrow TRUE$
    while $condtion == TRUE$ do
        for $index1 \leftarrow 0$ to length($causalGraph.relationships$) do
            $beforeState \leftarrow newState$
            if $causalGraph.relationships[index1] ==$ "$Proportional$" then
                applyProportionalRelationship($newState$)
            if $causalGraph.relationships[index1] ==$ "$Equivalence$" then
                applyEquivalenceRelationship($newState$)
            $condition \leftarrow ifDifferent(beforeState, newState)$
    return $newState$
```

- Checks for proportional relationships.
- Checks for equivalence relationships.

Algorithm 5: applyIntervalChanges(currentState,causalGraph)

- This method indicates the changes that can happen over a time interval.
- This method applies the influence relationship and derivative changes and records all possible state transitions from the *currentState*
- This method also applies exogenous change corresponding to the type of exogenous function in the *causalGraph* by updating the derivative of the inflow.

5 Results

Which deviations from the benchmark simulations will our modelling choices incur? And do they affect the overall degree to which our algorithm 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. Consistently with the Garp3 simulations we found that the state graphs of Model I and Model II coincide. Our algorithm also resulted in the same end state as the Garp3 implementation, showing that it captured the overall system behaviour well.

However, given the comparative simplicity of our algorithm, comparing our results with the benchmarks yields the following divergences:

- **States 9 and 10-17:** Here δ Inflow is negative although the inflow has already reached tero. This is excluded by the Garp3 modelling assumption “Apply quantity space constraints on zero as extreme Value” which we did not implement. However, it has no effects in this model.

Algorithm 5 Applying Interval Changes

```
procedure APPLYSTATICCHANGES(currentState,causalGraph)
    $newState \leftarrow currentState$
    $newStatesList[\,]$
    $counter = 0$
    for $index1 \leftarrow 0$ to length($causalGraph.relationships$) do
        if $causalGraph.relationships[index1] == "Influence"$ then
            applyInfluenceRelationship(newState)
        if ifDifferent($nextState, currentState$) then
            $newStatesList[counter] \leftarrow newState$
            $counter++$
    $newState \leftarrow currentState$
    for $index1 \leftarrow 0$ to length($causalGraph.entities$) do
        for $index2 \leftarrow 0$ to length($causalGraph.entities[index1].quantaties$) do
            applyDerivative($newState[index1][index2]$)
            if ifDifferent($nextState, currentState$) then
                $newStatesList[counter] \leftarrow newState$
                $counter++$
    $newState \leftarrow currentState$
    $newState \leftarrow$ applyExogenous($causalgraph.typeOfExogenous, newState$)
    if ifDifferent($nextState, currentState$) then
        $newStatesList[counter] \leftarrow newState$
        $counter++$
    return $newStatesList$
```

- **States 11-13:** These states do not appear in the Garp3 state graph. The reason is that state 11 is spurious: we have zero inflow and yet maximum volume and positive δ Volume. This happens since our algorithm does not check for second round effects of influence changes: when Inflow goes to zero this has the immediate second round effect of its influence on δ Volume becoming 0 thus being outweighed by I^-(Outflow, Volume) chaning δ Volume to 0 (third round effects abound). Instead, our algorithm only applies these second and third round effects step by step in states 12-14 which deterministically follow from 11.

- **State 15:** The same holds for state 15. Note that state 13 in figure 7 has a positve δ^2 Volume. This indicates the possibility to go to an equivalent to state 15 of our algorithm. However, this state is inconsistent to Garp3 since the immediate second round effect of I^-(Outflow, Volume) prevents transition to this state. Again, this is instead done step by step in two states by our algorithm.

- **State 16:** For the same reason, the algorithm takes two steps to transition from state 15 to state 17 resulting in the intermediary state 16

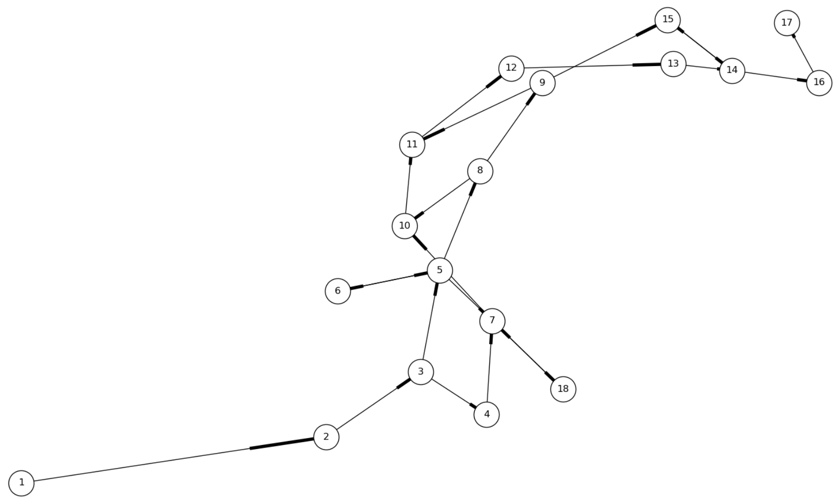


Figure 8: State graph for Model I and II according to our simulation.

6 Conclusions

In this project we implemented two models of a simple container system. Using a qualitative reasoning algorithm we explored the time evolution of these models. We found that our algorithm's state graph was well in line with intuition as well as with simulations of more advanced QR-software. In future implementations we would use an event based approach to deal with the immediate termination of second and third round effects.

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

Table 1: State table of Model I & II (right of children)