Modeling Hysteresis in CLIP – The Tank Flow Problem

David K. Wittenberg (dkw@cs.brandeis.edu) and Timothy J. Hickey
(tim@cs.brandeis.edu)
Brandeis University Department of Computer Science

Abstract.

Hickey and Wittenberg (Hickey and Wittenberg, 2004) study the "Two Tanks Problem", a hybrid system described by Stursberg *et al.* (Stursberg et al., 1997). In this paper, we expand on the use of CLIP (Hickey, 2000) (a Constraint Logic Programming over Intervals and Functions language) to formally describe more complex systems. We add complexity in several forms. The simplest is to have a larger system. We move from a system with two tanks to one with four tanks, and we add non-linear valves to the pipes connecting the tanks. This example easily generalizes to an N-tanks problem where the tanks, connected by pipes, form an arbitrarily complex graph. The more important addition is the refinement of the model in several places. We rigorously model a valve in which the flow varies exponentially with the valve position over much of the valve's range, and then discontinuously as the valve is almost closed. We introduce hysteresis in our analysis to avoid an infinite loop of zero-time transitions, and we discuss why our techniques should not have trouble with "Zeno" transitions.

The possibility of Zeno behaviour (Zhang et al., 2001) can arise either from physical reasons (a value near zero, so the sign of the changes is hard to know) or for modeling reasons (the system is near the boundary between two behaviour regimes, and while both regimes describe similar behaviour near the boundary, the model might switch between the two regimes infinitely often in a finite time). An elegant feature of our model is that we use the same technique of hysteresis to prevent the Zeno behaviour from either cause. This is easily done in CLIP by changing the conditions for a state change from one to the other to include hysteresis.

Keywords: Hybrid Systems, CLP, Intervals, Interval Arithmetic

1. Introduction

We use CLIP (a CLP language over analytic functions) to rigorously model hybrid systems. This paper extends our earlier work by using hysteresis to preserve the rigor of the model in the face of both non-analytic points and so called "Zeno" behaviour of a model.

There are two reasons for a point in the family of ODEs which describe a system to be non-analytic. One is simply that the ODE is non-analytic, and the other is that the system changes from a regime in which one ODE applies to a regime in which a different ODE applies. In a hybrid system, an ODE change can occur either because of a state change (the digital controller changes state) or because of what we call a "regime change" which is a point in which the system evolves from one regime to another - perhaps because a level passes a critical point.

A separate problem for hybrid systems is what are called “Zeno” systems. A Zeno system is one in which the model makes an infinite number of state changes in a finite amount of simulated time. This obviously causes the model to fail.

1.1. Hybrid Systems

A *Hybrid System* is a system composed of a digital part (typically a small computer) and an analog part (typically a physical system with sensors and actuators). The field of hybrid systems is the study of systems in which discrete events and continuous dynamic events interact. All computer controlled or monitored processes in the real world are hybrid systems. As a field of study, “Hybrid Systems” has come to include the study of the analog part of a system in an area where reliability is at a premium, typically because of the cost (in lives or money) of a failure of such a system. Some hybrid systems papers study only the analysis of the analog part of a system. Hybrid systems research grew out of real time computation, control theory, and program verification. Hybrid systems research strives to prove properties such as stability about complex safety critical systems. In the chemical engineering literature, hybrid systems are sometimes called “combined discrete/continuous processes”. An important use of hybrid systems is to prove “safety properties”, which are statements of the form “measurement x is within range $[a, b]$ such as “the water level in this tank never overflows”. Because safety properties are constraints, they fit naturally in a CLP approach. One widely studied hybrid system is the tank flow problem introduced by Kowalewski *et al.* (Kowalewski et al., 1999). It is this system that we discuss here.

The history of hybrid systems starts with Fahrland’s 1970 paper (Fahrland, 1970) which asked “Why limit the modeling to either discrete event or continuous when situations are evolving that require more interdisciplinary solutions”. Very little was done for the next twenty years, and Fahrland’s work is rarely cited. Fahrland may have been influenced by Roger Brockett who was also at Case Institute of Technology, and who later did some seminal work on hybrid systems. The first conference on the subject was the 1991 REX workshop titled Real Time: Theory in Practice (de Bakker et al., 1991) where the term “hybrid automata” was introduced. Since that time, real time systems and hybrid systems work has diverged, with real time work focusing more on the computer with its latency issues, and hybrid systems focusing more on accurate modeling of the analog part of the system. While it’s not clear how to put a real time model (explicit limits on the time for processing) in the standard formalism for hybrid automata, the techniques introduced in this thesis can easily model digital components as long as their latency can be bounded.

There has been considerable research on developing formal models of hybrid systems. Among others, Davoren and Nerode developed logics (Davoren and Nerode, 2000), Maler *et al.* (Maler et al., 1991), Lynch *et al.* (Lynch et al., 1999; Lynch et al., 2001), Henzinger *et al.* (Henzinger, 1996), and Alur *et al.* (Alur et al., 1995) developed formal models. From our point of view, a limitation of these models is the difficulty in applying them to real systems, and the amount of overhead that must be relied on to trust the results.

1.2. Earlier Interval and CLP Approaches to Hybrid Systems

We are not the first to apply interval arithmetic techniques to the problem of rigorously modeling hybrid systems. HyperTech (Henzinger et al., 2000) took a major step towards reliability of their results by using interval arithmetic ODE solving as a tool to add rigor to the very successful HyTech system. Our system merges, for the first time, the rigor of the formal model approaches and the practicality of the more engineering-based approaches by employing validated ODE solving. Our approach has several advantages over earlier interval models:

- CLIP is declarative, so that it describes the system being modeled directly.
- CLIP is logic based, so it can be viewed directly as a theorem prover using CLP logic.
- CLIP is constraint based. It doesn't require one to fully specify a system. CLIP allows one to understand some properties of a system based on initial assumptions.

Others have used constraint logic programming to model and analyze hybrid systems. Gupta *et al.* (Gupta et al., 1995)(Gupta et al., 1996) introduced a ground breaking approach called "hybrid cc" which allowed one to formally describe hybrid systems using a logic programming language with constraints. Urbina (Urbina, 1996) has pioneered another approach using CLP($\mathcal{R}$)(Jaffar et al., 1992) to model and analyze hybrid systems. Delzanno and Podelski (Delzanno and Podelski, 1999; Delzanno and Podelski, 2001) have explored analyzing hybrid systems using CLP(Q,R) (Holzbaur, 1995), a system which handles linear constraints with real and/or rational coefficients, as well as Boolean constraints. Their approach is to define a translator from Shankar's guarded command language (Shankar, 1993) to CLP(Q,R).

2. CLIP

CLP(I) is an interval-based constraint logic programming (CLP) language whose domain is the set of real numbers. The class of CLP languages (and their syntax and semantics) was introduced by Jaffar and Lassez in 1987 (Jaffar and Lassez, 1987). Jaffar and Maher provide an excellent survey (Jaffar and Maher, 1994) of the fundamental concepts of CLP. The idea of calculating over intervals of reals comes from Moore's 1966 book on Interval Arithmetic (Moore, 1966). The idea of combining CLP and Interval Arithmetic was first conceived by Cleary (Cleary, 1987) but the first production quality CLP(I) interpreter was the BNR Prolog system developed by Older, Vellino, and Benhamou (Research, 1988), (Benhamou and Older, 1997),(Older and Vellino, 1993). BNR Prolog was designed to be verifiably correct in the sense that the intervals it returned were mathematically guaranteed to contain all solutions to the underlying arithmetic constraints. The system however was proprietary and the underlying algorithms were never published in the scientific literature.

CLIP was originally developed as an open source implementation of CLP(I) by Qun Ju and Tim Hickey (Hickey and Ju,) (Hickey and Ju., 1997) CLIP has subsequently been extended by Tim Hickey, who added the CLP(F) language, which provides constraints over functions. CLIP is built on top of Prolog (Prolog 95, 1995), (Deransart et al., 1996), and currently runs on GNU Prolog (Diaz, 2002) and ALS Prolog. The fundamental philosophy is to have a relatively small base of sound primitive constraint contractors which are simple enough so that one can argue convincingly, if not formally prove, that they are correct, and then build more complex solvers on top of the proven system. Since the complex solvers built on CLIP primitives are made up of sound simple solvers, they are also sound. An important feature of CLP languages is that they are theorem provers, so that each answer generated by a CLP program has a direct interpretation as a theorem about the underlying domain.

CLIP can be considered to be a constraint engine over intervals and functions which interfaces to the Prolog engine (a constraint solver over general finite domains). The CLP(F) language solves analytic constraints by soundly approximating sufficiently differentiable functions by power series with remainder terms and introducing arithmetic constraints among the Taylor coefficients of the functions at the endpoints, at points in the interval, and over the entire range.

3. Generalized Tank Flow Problem

In a hybrid system, the interface between the analog and the digital part involves imperfect hardware whose description must include error bars. The models of system behaviour are often particularly imprecise near boundaries. We use intervals to handle the issue of imprecision in measurements, and use intervals in a novel way to rigorously model the behaviour of systems near boundary points. We start by adding valves to the model. We note that the model in Kowalewski *et al.* has the behaviour of the valve discontinuous at 0 (by 5% of full flow), and show how a broad constraint describes that.

In (Wittenberg, 2004) we showed how CLIP could model the simple two tanks problem. In this paper, we show how the CLIP model can easily be extended to the "tank flow problem", an extension of the two tanks system to an arbitrary number of tanks, and to model it more rigorously than other methods can. Here, we consider a four tank version with valves between each pair of tanks and at the output.

3.1. Mathematics of the Tank Flow Problem

The problem we study is diagrammed in Fig. 1 and the parameters and variables are shown in Table I. The problem can be described as follows: There are n tanks, numbered from 1 to n, with the bottom of each tank lower than the bottom of the previous tank. The depth of the water in tank j at time t is given by $D_j(t)$ The depths D_j are measured from the bottom of their respective tanks. The altitude of the bottom of tank j is H_j above an arbitrary horizontal datum, perhaps sea level. Each tank j has a horizontal pipe leaving from the bottom of the tank. The flow through that pipe is I_j, and there is a valve V_j on the pipe. There is a constant inflow of water into tank 1 (the uppermost tank) where the flow rate is given by a constant $f00$.

The general equation for flow through a pipe is that the rate of flow is proportional to pipe coefficient times the square root of the height difference of the water levels at each end. Specifically, the flow $I_j(t)$ through pipe j connecting tank j to tank $j+1$ is governed by a pair of ODEs in the resistance $R_j(t)$ to flow.

$R_j(t)$ is a function of the pipe coefficient C_j, valve coefficient E_j, and the valve position $P_j(t)$) and to the square root of the pressure difference. The pipe coefficient C_j describes how easily water flows through the pipe when the valve is in the fully open position. The valve coefficient E_j is the exponent describing how much the valve cuts off the flow as a function of the valve position. The pressure difference is proportional to the difference in water heights on each end of the pipe.

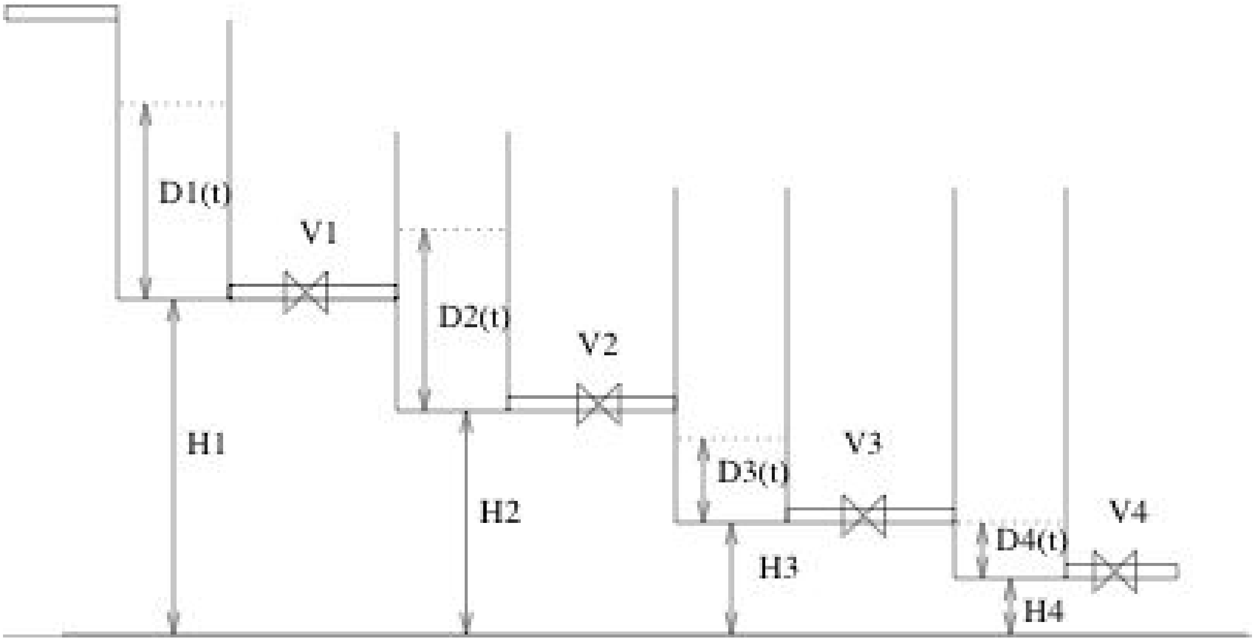

Figure 1. Diagram of Tank Flow system for $n = 4$

Table I. Parameters and Variables

H_j	Height of tank j above sea level
V_j	inverse of time for valve j to open or close (valve speed)
C_j	pipe coefficient of pipe j when the valve is fully open
E_j	exponent for describing the valve's behaviour
$P_j(t)$	position of valve j. 0 is fully closed, 1 is fully open
$M_j(t)$	valve motion – `closing, opening, halted`
$R_j(t)$	program variable for valve regime - `shut, transition, normal`
$D_j(t)$	Depth of water in tank j at time t (measured from bottom of tank.)
$I_j(t)$	rate of flow through pipe j at time t

If the water level in the lower tank is below the pipe bringing water in, there is no back pressure in the pipe, so we can ignore the water level in the lower tank. If the water level in the lower tank is higher than the input pipe, we have to include the effect of back pressure on the flow through the pipe. Therefore, we have a pair of ODEs for each pipe. One ODE of the pair holds when the water in the lower tank $(j+1)$ is below the level of the connecting pipe $(D_{j+1}(t) < H_j - H_{j+1})$, the other member of the pair holds when the water level is above the connecting pipe $(D_{j+1}(t) > H_j - H_{j+1})$. When the water level is equal to the height of the connecting pipe, the ODEs are the same, so we choose one arbitrarily. Later (Section 5.2) we will show how to rigorously handle this point where the ODEs change, and which is therefore not analytic. Note that even if the water level is above both ends of a pipe, if the water levels (measured from sea level) are equal, the ODE is non-analytic because the square root[1] function's derivative is infinite at 0.

In our simulation, later tanks do not have higher water level than earlier tanks, so we do not consider "backwards" flow, though it is a simple extension.

We handle these two different reasons for an ODE to be non-analytic in exactly the same manner, described in Section 5.

The valve decreases the flow by a fraction which decreases exponentially with the valve position. Recall that since D_j is the depth in a tank, I_{j-1} the flow into that tank, and I_j the flow out, $D'_j = I_{j-1} - I_j$ Define HDiff_j to be the the difference in altitude between the bottom of tank j and the bottom of tank $j+1$ That is: $\mathrm{HDiff}_j = H_j - H_{j+1}$ The ODEs for flows in pipe j are:

$$I_j(t) =$$

$$\begin{cases} 0 & P_j(t) = 0 \\ e^{E_j \cdot (1-P_j(t))^3} \cdot C_j \sqrt{D_j(t) - D_{j+1}(t) + \mathrm{HDiff}_j} & D_{j+1}(t) > \mathrm{HDiff}_j \\ e^{E_j \cdot (1-P_j(t))^3} \cdot C_j \sqrt{D_j(t)} & D_{j+1}(t) \leq \mathrm{HDiff}_j \end{cases}$$

Where P_j is the position of the valve; C_j is the pipe coefficient; the value under the radical is the effective difference in height between the water levels of the two tanks, and the exponential term is the fraction by which the valve decreases the flow.

4. Handling State Changes

A hybrid system of any size will have different ODEs to describe it at different times. Writing each ODE explicitly (as we did for a simpler example in (Hickey and Wittenberg, 2004)) is impractical because of a combinatorial explosion in the number of ODEs. To avoid this problem, we parameterize the ODEs describing the system, so a state change is modeled by a change in some of the parameters to an ODE rather than by making a different ODE active.

The ODEs governing a hybrid system can change for either of two reasons. The first is if the digital part of the system has a state change which affects the ODEs. We call this a *program control change.* The other is if the continuous system evolves in such a way as to change the ODEs, such as evolving to a point where a tank overflows, or the water level in a tank rises above the input pipe to that tank, causing back pressure. We call these events *regime changes.* One case of regime change is when a valve that had been opening (or closing) becomes fully open (or closed). That affects the ODEs, by changing the rate at which the valve position changes, not by changing the water flow directly. A helpful feature of CLP(F) is that we can model changes in ODEs caused by program control and those caused by regime changes in exactly the same way.

[1] We really want a function which is the positive square root of a positive number, and the negative square root of the absolute value of a negative number to properly describe the fluid flow. This function is also not analytic at 0.

Figure 2 is a state diagram for each valve except the last in the tank flow problem. (The last valve has no lower tank, so the level in the lower tank can't rise above the height of the pipe.) The states are described by two ternary variables, M (valve motion regime) describes the motion of the valve as one of (`opening, closing, halted`), while R (valve position regime) is one of (`shut, trans, normal`). When R takes the value `shut` it means that the valve is closed, `normal` means that the valve is open, and not too near the closed position. When R takes value `trans` the valve is in a transitional region and is nearly, but not quite closed. The transitional region is used to model the regime where the ODEs are not well understood, so we use a simple over-approximation constraint in that regime.

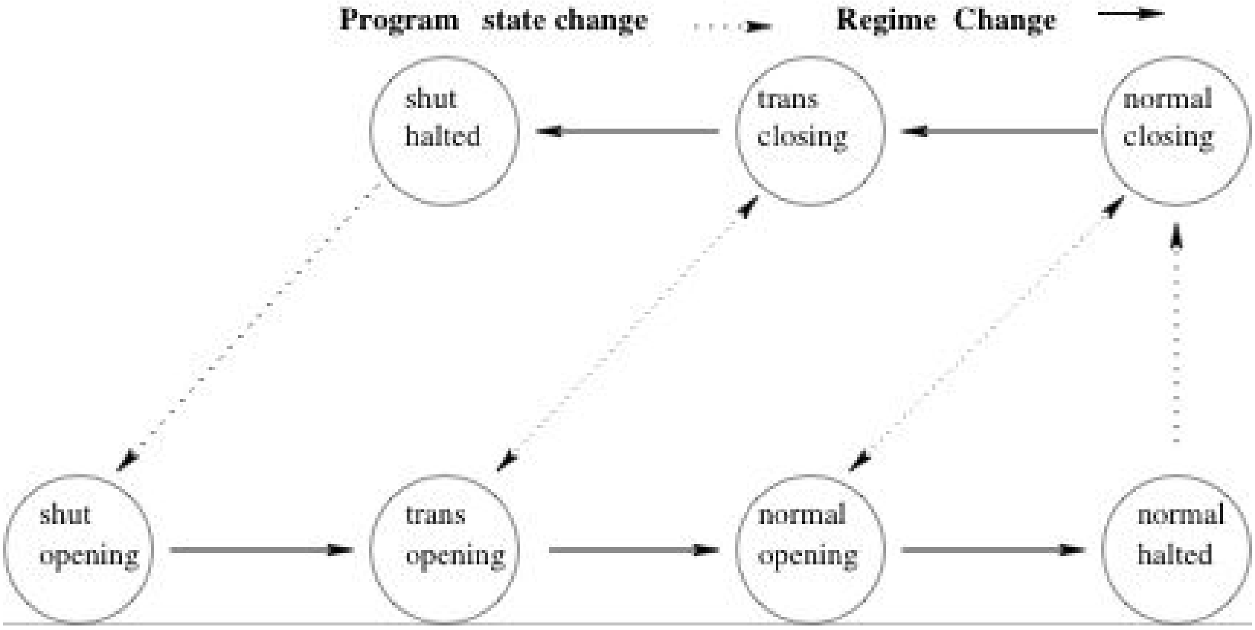

Figure 2. State diagram for ODEs

5. Unavoidable Sources of Error

An important issue in modeling hybrid systems is to realize that almost none of the parameters are known exactly. This is true both of the parameters to the differential equations which describe the system – These parameters are often determined by curve fitting to a set of measured points or are calculated from physical models which include simplifying abstractions, and of measurements taken by sensors in the system – These are measured with some accuracy, which is often specified as an error-bar.[2] Because CLP(F) treats everything as an interval, it models these error bars naturally. If a CLP(F) program shows that a system has a safety property (proves that it avoids a region), that proof is valid even when each parameter takes the worst possible value within the given error bars. To deal with this issue by sensitivity analysis (sensitivity analysis conference, 2004; Arsham, ; Taylor, 1997) on the inputs would be extremely difficult.

One of the problems in rigorous modeling is that often there are areas where one's original model breaks down for some reason. This can occur at an area where the physics are unclear, a point where the defining functions are not analytic, or perhaps a function which is poorly defined at a limit point. The point of this paper is that CLIP makes it simple to use hysteresis to deal with all of these problems.

5.1. Dealing with Poorly Defined Regions

Ideally a modeling system allows stepwise refinement of the model. We demonstrate this in CLP(F) by adding valves to our model of the tank flow system. Adding the valves to the model was easy despite using a rather complex model of the valve's behaviour.

One problem which is rarely addressed in modeling hybrid systems is modeling the area around where a component or valve changes state. Using constraints, we can provide a rigorous answer by describing the output of the component while it changes state as being between the output it has in one state and the output it has in the other, and keeping that constraint for however long the component takes to change state. If more precision is required, one can add a description of the component's behaviour during the state transition. Since the description consists of upper and lower bounds for the component's output, one can progressively refine the bounds as one learns more about the component's behaviour.

In many systems, the physics in some regions is not well-understood. Most hybrid system techniques ignore this and simply assume that the ODEs which work in most areas work near boundaries as well. For example, in the tank flow problem when a tank is almost empty, the flow from it may be irregular and come in discrete drops rather than as a continuous flow. At these points, we don't claim to understand the details of the flow, but we can model them rigorously by writing constraints which clearly include any possible behaviour of the flow. We don't consider an empty tank in this case, except to constrain our description of the system to cases in which the water level in each tank is at least `E`, where `E` is a negligibly small positive value.

A further problem is that even away from boundary conditions, the physics of the system may not be understood perfectly. In most cases, one measures a value (here, the flow through a valve as a function of how open the valve is) at several points, uses physical theory to decide what form the curve should be (in this case, an exponential of the valve position), and then uses a least-squares fit to find a curve which best describes the measurements. There is, of course, error in the measurement of each point, so the coefficients for the exponential curve have some (hard to calculate) error bars. In addition, the behaviour when the valve is almost closed does not follow the exponential decay curve, and is extremely difficult to measure precisely.

[2] Note that the problem of imperfect measurement is inherent in the physical world. Heisenberg's uncertainty principle prohibits perfect measurement, and Burridan's principle (Lamport, 1986) further limits the speed at which one can usefully take measurements.

For example Kowalewski *et al.* (Kowalewski et al., 1999) describe a valve by a function $K_i(P)$ giving the pipe coefficient and valve coefficient of the valve as an observed function of how open the valve is. For the first valve, the function they give (converted to our notation) is:

$$K_1(P_1) = \begin{cases} 1.85 \cdot 10^{-4} \cdot e^{-3.1 \cdot 1 - P_1^3} \frac{m^{5/2}}{s} & \text{if } 0 < P_1 \leq 1 \\ 0 \frac{m^{5/2}}{s} & \text{if } P_1 = 0 \end{cases}$$

and for the second valve, they give:

$$K_2(P_2) = \begin{cases} 2.26 \cdot 10^{-4} \cdot e^{-5.7 \cdot 1 - P_2^3} \frac{m^{5/2}}{s} & \text{if } 0 < P_2 \leq 80 \\ 0 \frac{m^{5/2}}{s} & \text{if } P_2 = 0 \end{cases}$$

In neither case do they give error bars. The valve position is described by a real number in $[0, 1]$ with 0 corresponding to fully closed and 1 to fully open. Figure 3 shows a graph of R vs. P for valve 1. The curve is an exponential decay, whose value when the valve is almost closed is about 5% of the flow when the valve is wide open, but they define the flow for a fully closed valve as 0. By straightforward calculation, we find that $R_1(1) \approx 1.85 \cdot 10^{-4}$, while $R_1(\varepsilon) \approx 8.570 \cdot 10^{-6}$ (this is about 5% of full flow), and $R_1(0)$ is defined to be zero. It is likely that this is not fully correct, as a discontinuity of that magnitude is not common. We model this discontinuous point by having three constraints for three different regimes. When the valve is fully closed, R is 0. When the valve position is above the transition region, R is given by the ODE above. The interesting case is when the valve position is in the transition region. We model this case with a constraint which says that if the valve position P is near 0 (here we specify < 0.02), R is small. To choose the upper end of R's range, we choose a value slightly above the calculated value of R at any point in P's range for that region. The choice of where the transition region ends is somewhat arbitrary.

Figure 4 shows how we rigorously model this system for P near 0. For the part of the curve where the equations are reliable, we enclose the specified curve on each side by the ODE describing the valve. Because the parameters of the ODE are intervals, the value of the function at any point is an interval. In the area where the curve is discontinuous, we use a constraint which includes all possible values the function could take. This introduces some uncertainty into the formal model, but that uncertainty was already present in the description of the physical system. Using constraints makes that uncertainty explicit, and models it rigorously.

5.2. Dealing with Regime Change Points

One of the advantages of using CLP(F) is that one can often use one technique to handle multiple issues. In section 5.1 we use separate ODEs, often with rather simple-minded constraints, to deal with regions where the physics is unclear. Here we use a similar system to deal with non-analytic (or even discontinuous) points in an ODE.

When the water level in the lower tank is above the input pipe (in regime `above`), one set of ODEs holds, when the level is below the input pipe (in regime `below`), another set of ODEs holds. We model this by having a regime change at that point. An obvious problem arises: Our model would allow an infinite number of transitions (each taking zero time) between the two states, and therefore never get to calculating the change in water level which would move clearly into one state

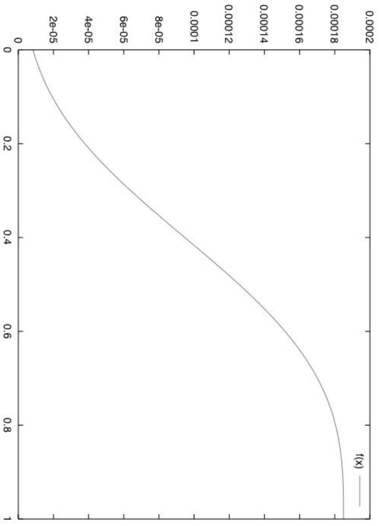

Figure 3. Relative flow as a function of valve position for valve 1 – function from Kowalewski *et al.*

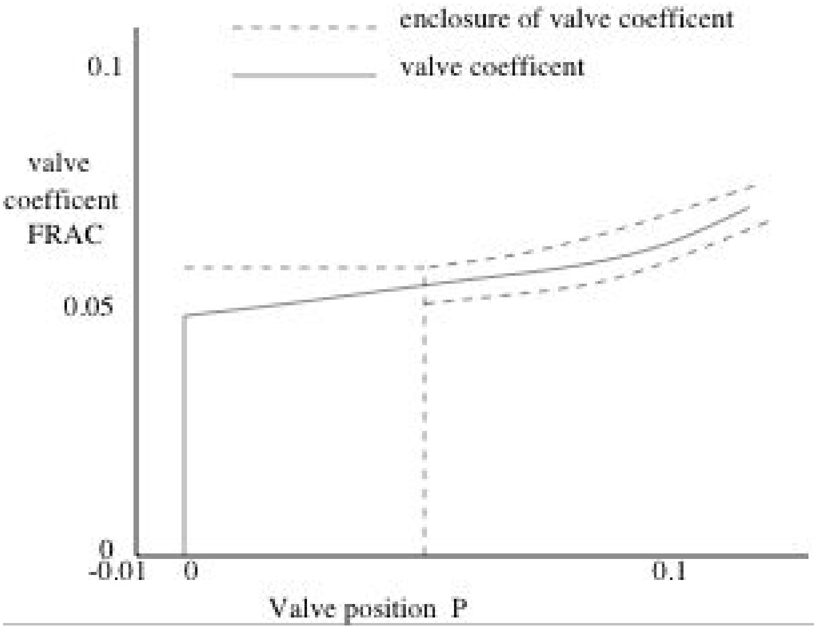

Figure 4. Graph of flow against valve position for valve 1 showing enclosure by simple constraint (scale greatly enlarged)

(Actually, since CLPs are non-deterministic, there would be an infinite path and also a one-step path out of that condition). We handle this by creating a special artificial state `near` for water levels near the boundary. We then artificially put in hysteresis, so that on leaving that middle state, one cannot immediately re-enter it. This problem is related to the problem of Zeno automata, discussed in Section 7.

This example clarifies two issues, as there are two separate reasons for using `near` state between `above` and `below`. The first reason is that as the water in the lower tank reaches the level of the pipe the physics get a little unclear - what happens when the water covers half the pipe? This issue is clearer in the case where the ODEs are discontinuous, as in section 5.1. The second issue is that in order to model a change of ODEs, we need two regimes, with appropriate transitions between them. This issue arises even when the physics are clear, such as when one has a pipe between two tanks and the relative water levels in the two tanks is changing. At the point where the water levels are equal, the ODE is non-analytic (because the square root function is non-analytic at 0), so we would have to have a change of regimes. If the rule for a regime change was simply that the water levels were equal, when the levels became equal there would be a legal infinite path of zero-time changes from one regime to the other. One could look at the derivative to know which direction the regime change goes in, but if the water level is almost constant, the derivative will be near zero, and the same issue is still there. To avoid this case, we artificially add hysteresis to an already artificial regime change.

6. Overview of Code for Tank Flow

The complete program for the n=4 case of the tank-flow problem is in the Appendix of(Wittenberg, 2004). Here we discuss some of the more interesting snippets from the code.

6.1. Evolve and Iterate

We model a hybrid system in CLP(F) by modeling a series of steps. A step begins either at a specified initial state, or when the previous step ends, and ends when either the length of the step (amount of time simulated) reaches a maximum step size `delta`, or a change of ODEs occurs (whether caused by program control or a regime change). The following part of the program is the main code, which runs the system through one step, increments the state counter, and continues.

```
evolve(S0,C,N,S2) :-
  evolve0(S0,C,N,S1),
  enforce_ODEs(S1,C,S2),
  copy_discrete_state(S1,S2).
```

`evolve(S0,C,N,S2)` is true if and only if the system described by `C` can evolve to a boundary state `S1` in `N` steps and then evolve from state `S1` to state `S2`.

```
evolve0(S0,_C,N,S1) :- {N=0},eqstate(S0,S1).
```

```
evolve0(S0,C,N,S2) :-
    opt_next_step(S0,C,S1),
    print(ons(S0,C,S1)),nl,nl,
    {N=M+1},
    evolve0(S1,C,M,S2).
```

A direct reading of the program is as follows: In zero steps, the system does not change state. The `evolve0` predicate says that a system can evolve from `S0` to `S2` if `S1` is the next step from `S0`, $N = M + 1$, and the system can evolve from `S1` to `S2` in `M` steps. The variable `C` in all cases is the set of constants which describe the system parameters.

```
% next_step(InitialState, ProblemConstants, FinalState)
next_step(S0,C,S1) :-
  enforce_ODEs(S0,C,S1),
  find_state_change(S0,C,S1).
```

The call to `next_step` states that the ODEs are followed (`enforce_ODEs`), and finally that system has run to an appropriate point (`find_state_change`). All the variables in the states (`S0, S1`) and the constants term (`C`) are variables over the reals. Variables over functions are used in `enforce_ODEs` to specify constraints over the real variables in `S0, C, S1`.

6.2. Finding State or Regime Changes

In each case, the step ends when any of the requirements becomes true. Figure 5, shows how `find_state_change` is defined to be true when any one of the following happen:

- One of the `find_flow_state_change` predicates becomes true because the water level in one of the tanks goes from above the input pipe in state `S0` to below in state `S1`, or vice versa (one of the tanks changes regime)
- one of the `find_valve_state_change` predicates becomes true, because the valve position is such that a change in regime occurs at state `S1`
- one of the `find_program_state_change` predicates becomes true because the program (ie. the digital part of the hybrid system) changes state at state `S1`
- `find_step_change` is true because state `S1` is `Delta` time after state `S0` and no other state changes have occurred.

The CLP(F) code to check for this (excerpted in Figure 5 looks rather repetitive. This is true only because in this example we use the same behaviour for each valve and for each tank. In a less symmetric case, this code would not grow, but there might have to be multiple versions of `find_??_state_change` to describe the different behaviours.

```
find_state_change(S0,C,S1) :-
% TANK FLOW REGIME CHANGE
         find_flow_state_change(r1,h1,h2,d2,C,S0,S1);
         find_flow_state_change(r2,h2,h3,d3,C,S0,S1);
         find_flow_state_change(r3,h3,h4,d4,C,S0,S1);

% VALVE REGIME CHANGE
          find_valve_state_change(p1,vr1,vm1,C,S0,S1);
          find_valve_state_change(p2,vr2,vm2,C,S0,S1);
          find_valve_state_change(p3,vr3,vm3,C,S0,S1);
          find_valve_state_change(p4,vr4,vm4,C,S0,S1);

% PROGRAMMED STATE CHANGES
          find_program_state_change(v1,d2,C,S0,S1);
          find_program_state_change(v2,d3,C,S0,S1);
          find_program_state_change(v3,d4,C,S0,S1);

%  no regime or state changes before the time limit is reached
          find_step_size_change(S0,C,S1).
```

Figure 5. Code to Find State Changes

6.3. Enforcing ODEs

One section describes all of the analog parts of the system. It consists of three large assertions. The first (and largest) section is purely bookkeeping. All of the ODEs are in the last two parts of `enforce_ODEs`. In order to make the lists of parameters smaller, we use lists to keep all variables of each type together. `lookup`, `evalall`, and `decls` are helper functions to deal with the lists.

The bookkeeping section states that the individual variables correspond to what the lists say they are, and constrains the domain and range of the functions. It uses `lookup` to bind the values of constants (from `C`), and conditions at the start of the step (from `S0`), and the end of the step (from `S1`) to variables. Then it uses `decls` to declare several function variables (and their domains) at once, and finally specifies which ODEs each tank should obey while in the state specified by `S0`. Figure 6 shows sections of the first part of `enforce_ODEs`. Much of that section is repetitive, so only representative fragments are reproduced here. We interpret the code as follows: `enforce_ODEs` is true if and only if all of the following elements are true (including, of course, those that are elided here.)

- `P` is a vector containing `P1,P2,P3,P4`
- each element of `P` is a function defined on $[\mathrm{T0}, \mathrm{T1max}]$
- each element of `P` is a function whose range is $[0, 1]$

- Ps0 is a vector containing P10,P20,P30,P40
- Ps1 is a vector containing P11,P21,P31,P41
- the values of Ps0 are as specified in C in state S0
- the values of Ps1 are as specified in C in state S1
- evalall applied to each element in the list P evaluated at T0 gives the corresponding value from Ps0, and when evaluated at T1 gives the corresponding value from Ps1
- the value of v in the list of constants C is V
- each of the valves obeys valve_ODE given the valve position, velocity, and motion regime
- each of the tanks obeys the appropriate tank ODE

Another section handles the flow restriction caused by the valves.

```
valve_coef(normal,FRAC,P,E) :- {[FRAC=exp(E*((1-P))**3),
                               FRAC in [0,1],  P in [0.01,1] ]}.
valve_coef(trans,FRAC,P,_) :- {[ FRAC in [0,0.06],  P in [0,0.01] ]}.
valve_coef(shut,FRAC,P,_) :- {[FRAC=0.0*FRAC, P=0*P ]}.

valve_ODE(P,_,halted) :- {[ ddt(P,1) =   0.0*P, P in [0,1] ]}.
valve_ODE(P,V,opening) :- {[ ddt(P,1) =  V+0*P, P in [0,1] ]}.
valve_ODE(P,V,closing) :- {[ ddt(P,1) = NV + 0*P, P in [0,1],
                                                  NV= - V ]}.
```

The ODE code is completely straightforward, as ODEs can be described directly in CLP(F). FRAC, E, and P are all functions of T. The first line says that in the valve regime normal,

$$\mathrm{FRAC} = e^{E\cdot(1-P)^3},\;\; \mathrm{FRAC} \in [0,1],\;\; P \in [0.01,1]$$

The second line says that in valve regime trans FRAC $\in [0, 0.06]$ and $P \in [0, 0.01]$. The third line says that flow through a shut valve is 0. The idiom FRAC=0.0*FRAC is a workaround used instead of FRAC=0 because CLIP does not allow functions to be set equal to a constant. The second line of the code is needed to implement the technique of rigorously modeling discontinuous functions discussed in section 5.1. Observe that this procedure constrains P to take values inside the appropriate region (for normal, trans, shut).

Similarly, the last three lines specify the derivative of P (the valve position) to be 0 when halted, V for opening, and -V for closing.

The last assertions in the ODE section specify the flow into and out of tanks. There are seven cases,as the first and last tanks have different configurations than tanks in the middle, and for all but the last tank, the ODEs differ according to which regime the tanks is (among below, near, and above) corresponding to whether the water level in the lower tank is above or below the pipe entering the lower tank. We consider the case of a middle tank in regime above, as that is the most complex.

```
enforce_ODEs(S0,C,S1) :-

...

  % VALVE Position ODEs
  % create valve position functions on [T0,T1max]
    P=[P1,P2,P3,P4],
    decls(P,function(T0,T1max)),
  % put bounds on the range of the function
    bound_functions(P,[0,1]),
  % set their values at times T0 and T1
    Ps0=[P10,P20,P30,P40],
    Ps1=[P11,P21,P31,P41],
    lookup([p1=P10,p2=P20,p3=P30,p4=P40],S0),
    lookup([p1=P11,p2=P21,p3=P31,p4=P41],S1),
    evalall(P,T0,Ps0), evalall(P,T1,Ps1),
  % lookup the valve speed
    lookup([v=V],C),
  % add the ODE constraints
    valve_ODE(P1,V,M1),
    valve_ODE(P2,V,M2),
    valve_ODE(P3,V,M3),
    valve_ODE(P4,V,M4),
...

  % apply the ODEs corresponding to each tank
  % Ri = ode governing tank i, Di = depth in tank i,
  % Fi = flow out of tank i, Hi = height of tank i,
  % Pi = valve opening out of tank i, Ki = valve coefficient,
  % F00 = flow into tank 1

        first_tank( R1,    D1,F1,D2,  F00,H1,C1,FRAC1,H2,E),
        middle_tank(R2, F1,D2,F2,D3,      H2,C2,FRAC2,H3,E),
        middle_tank(R3, F2,D3,F3,D4,      H3,C3,FRAC3,H4,E),
        last_tank(      F3,D4,F4,            C4,FRAC4).
```

Figure 6. Parts of Enforce ODEs code

```
middle_tank(above,F1,D2,F2,D3,H2,C2,FRAC2,H3,_E) :-
 {[F2=C2*FRAC2*psqrt(D2-D3+H), ddt(D2,1)=F1-F2, H=H2-H3,
                                                  D3 in [H,1000] ]}.
```

This says that given a middle tank 2 (middle tank here means that tank 2 is not the first tank, and tank 3 is not the last tank) in regime `above` with the following parameters:

`F1`	flow into the upper tank
`D2`	water height of the upper tank
`F2`	flow out of the upper tank (into the lower tank)
`D3`	water height of the lower tank
`H2`	height of the upper tank above sea level
`C2`	parameter of flow through the pipe between upper and lower tanks
`FRAC2`	fraction of the maximum flow the valve allows
`H3`	height of the lower tank above sea level
`_E`	an error term (the underscore before the `E` means ignore this term .)

then:

$$F2 = C2 \cdot \mathrm{FRAC2}\sqrt{\mathrm{D2} - \mathrm{D3} + \mathrm{H}},\ \ \frac{d\mathrm{D2}}{d\mathrm{T}} = \mathrm{F2} - \mathrm{F1},\ \ \mathrm{H} = \mathrm{H2} - \mathrm{H3},\ \ \mathrm{D3} \in [\mathrm{H}, 1000]$$

Here `H2, C2, H3` and `E` are constants, and all the other variables are function variables, though that must be implied from earlier declarations. Again note that the the constraint requires the depth `D3(T)` to be in the region for the `above` case or on the boundary with another case. Note how the ODEs translate directly into CLIP.

6.4. Finding State Changes

The last section of code we describe in detail determines that a regime change has occurred. Parts of this code are in Figure 7 and Figure 8. This code is called from `find_state_change` which says that `find_state_change` is true if at least one of `find_flow_state_change, find_valve_state_change, find_program_state_change` or `find_step_change`, is true.

`find_flow_state_change` (Figure 7) is true if and only if the two `lookup` assertions are true, `update_discrete_state` is satisfied, and `flow_state_change` is satisfied. The `lookup` assertions state that the values of constants passed to the assertion match the constants stored in `C`. `update_discrete_state` here states that the only difference in discrete variables between state `S0` and state `S1` is that in state `S0`, `Ri` has value `R_before` and in state `S1`, `Ri` has value `R_after`.

`flow_state_change` lists the four possible transitions, and the water levels which allow them. Note the hysteresis – to enter state `near` the water level has to be within `E` of the critical level (`H1 - H2`), while to leave state `near` the water level has to be `2*E` away from the critical level. This is to prevent an infinite sequence of zero-time transitions when the water level is at a critical point.

Figure 8 shows the code for changes in the valve's regime. `find_valve_state_change` is very similar to the code for `find_flow_state_change`, except that it twice calls `update_discrete_state` to update the two ternary variables for the two sets of regimes a valve has. One (`M`) is the valve

```
% Detection of regime change due to tank depth exceeding input
% pipe height  e.g. find_flow_state_change(r2,h2,h3,d3,C,S0,S1).
% note j=i+1 here and Ri in {above,near,below}

find_flow_state_change(Ri,Hi,Hj,Dj,C,S0,S1) :-
  lookup([Hi=H1,Hj=H2],C),  lookup([Dj=D],S1),
  update_discrete_state(Ri,R_before,R_after,S0,S1),
  flow_state_change(R_before,D,R_after,H1,H2).

% We use hysteresis in our analysis to avoid an infinite loop of zero
% time state changes as it goes from near to below and back again.
flow_state_change(below,D,near,H1,H2) :-
  E=0.00001,  {D = H1-H2-E}.
flow_state_change(near,D,below,H1,H2) :-
  E=0.00001,  {D = H1-H2-2*E}.
flow_state_change(above,D,near,H1,H2) :-
  E=0.00001,  {D = H1-H2+E}.
flow_state_change(near,D,below,H1,H2) :-
  E=0.00001,  {D = H1-H2+2*E}.
```

Figure 7. Code for Regime Change as Water Level Changes

motion regime, which can be one of `opening, halted, closing`, the other (`R`) is the valve position regime, which can be one of `shut, trans, norm`. The valve position regime is necessary because of the discontinuity in the valve ODEs at zero. `norm` means that the valve is in the regime where the standard ODE applies, `shut` means that the valve is fully closed, and there is no flow through it, and `trans` is the transition regime, where we simply apply a coarse constraint because we don't understand the physics in that regime.

7. Zeno hybrid systems

Johansson *et al.* (Johansson et al., 1999) introduce what they call a "Zeno phenomenon". This is a problem with some hybrid models in which an infinite number of steps occur in a finite amount of time. At best, this leads to calculations which never finish, while at worst, it leads to false proofs of safety properties in systems which don't have those properties. The canonical examples of Zeno phenomena are a bouncing ball which with each bounce achieves some fraction of the height of the previous bounce in a fixed fraction of the time, and a water tank example discussed below. In the bouncing ball case, a simulation would have to calculate an infinite number of bounces before terminating unless the model included some handling of the idea that when the height of each bounce is less than one atom's diameter, the model must change.

The water tanks example of Johansson *et al.* is shown in Figure 9. There is a flow of water i into a valve which can direct the water into either of two tanks. Each tank has a water level (h1, h2),

```
% check to see if valve n has hit a state change
% and if so, update the discrete part of S1 accordingly
% e.g. find_valve_state_change(p2,v2,C,S0,S1).
% v2 in {opening,closing,halted}, p2 in [0,1],
% note that this is a regime change, not a state change.
% Also, we have to handle the regime change from shut to transition
% to normal. The transisition to shut implies a transition to halted,
% but not vice versa.

find_valve_state_change(Pn,Rn,Mn,_C,S0,S1) :-
% use S2, as temp states to have 2 discreet vars change
  lookup([Pn=P_before],S0),lookup([Pn=P_after],S1),
  update_discrete_state(Mn,M_before,M_after,S0,S2),
  update_discrete_state(Rn,R_before,R_after,S2,S1),
  valve_state_change(M_before,R_before,P_before,M_after,
                                             R_after,P_after).


% regime change rules for valve motion (and in closing case, position)
valve_state_change(opening,normal,_P_before,halted,normal,P_after) :-
                                            {P_after=1}.
valve_state_change(closing,trans,_P_before,halted,shut,P_after) :-
                                              {P_after=0}.

% regime change rules for valves position
valve_state_change(opening,trans,_P_before, opening,normal,P_after) :-
                                                     {P_after=0.01}.
valve_state_change(opening,normal,_P_before,constant,normal,P_after):-
                                                     {P_after=1.0}.
valve_state_change(opening, shut,_P_before,  opening,trans,P_after) :-
                                                     {P_after=0.0}.
valve_state_change(closing, normal,_P_before,closing,trans,P_after) :-
                                                      {P_after=0.01}.
valve_state_change(closing,trans,_P_before,  constant,shut,P_after) :-
                                                       {P_after=0.0}.
```

Figure 8. Code for Valve Regime Changes

and a required level (r1, r2). The safety property is that h1 is always above r1, and h2 is always above r2. The water flow out of each tank is proportional to the ratio of the area of the tank to the area of the output pipe, and to the square root of the water height. If the input flow i is chosen to be larger than either output flow $o1$ or $o2$, but less than their sum (when h1 and h2 are near r1 and r2), it is clear that the level in at least one of the tanks must fall below its required level. Consider the program which whenever one of tanks gets to its required level switches the flow to that tank. As the water level gets lower, the switching will happen more and more often, and the valve will switch an infinite number of times in a finite period, during which time the water level in each tank will still be at or above the required level.

Johansson *et al.* note that the Zeno phenomenon usually occurs as a result of over abstraction in the model, as happens in these cases. Real systems can have valves that chatter, but the chattering cannot involve an infinite number of state changes in a finite time. If the real system has chatter, one should model it by a constraint giving a minimum time for a valve to change state. The infinite chattering is an artifact of some models, and should be removed by the modeler. Zhang *et al.* (Zhang et al., 2001) give examples of cases where overly abstract models (with the Zeno property) of real systems (without the Zeno property) lead to incorrect proofs of safety properties. In most cases, the Zeno problem can be eliminated by a more accurate model, often by simply modeling the time a valve or switch takes to change state. In our example, we avoid Zeno phenomena because there is a lower bound on the time required for twelve consecutive state changes. This bound is implied in different ways for different sets of state changes. For example, the water flow through any pipe is proportional to the square root of the water height, and we bound the water height in each tank. That limit on the water flow limits how quickly the water level in any tank can change. The only code added to avoid Zeno phenomena is the hysteresis. One case in which we do not avoid Zeno phenomena is if the discrete part of a hybrid automata describes a Zeno phenomena. If, for example, the program specified that at some water level a valve would switch from open to closed and from closed to open, that behaviour would be modeled, and the simulation might never finish. There is nothing to be done here. If a user specifies a poorly-formed program, analysis may fail.

7.1. CLP(F) and Zeno systems

How does a CLP(F) model handle a Zeno system? Consider the bouncing ball first. If the modeler does not note that the physics change for very small bounces, the simulation has to include an infinite number of vanishingly small bounces, but because everything in CLP(F) is an interval, the height of the bounce will at some point reduce to $[0, S]$, where S is the smallest number representable in the floating point system. CLP(F)'s non-determinism means that it should eventually explore the path where the bounce height is 0, and the motion ends. Because CLP(F) currently uses depth-first search, it is non-deterministic whether it will try the finite or the infinite path at each branch. If CLP(F) were to use breadth-first search, it would clearly show the possibility that the motion ended, while still modeling the Zeno execution as another possibility. This is probably the best one can hope for. If one gives a computer a model which includes a Zeno execution, the model must show that. If the model can also show that the behaviour is within measurement (or calculation) error, one hopes the user will realize that the initial model is insufficiently defined.

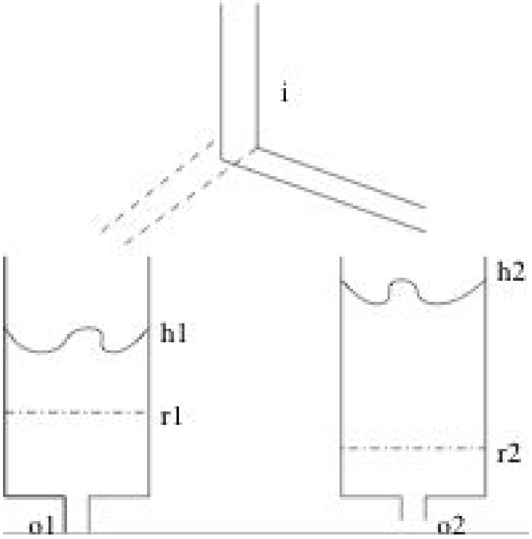

Figure 9. Flow system with Zeno behaviour (after Zhang *et al.*)

References

Alur, R., C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine: 1995, 'The Algorithmic Analysis of Hybrid Systems'. *Theoretical Computer Science* **138**, 3–34.

Arsham, H., 'Sensitivity Analysis'. A collection of recent developments on sensitivity analysis in several fields, http://ubmail.ubalt.edu/∼harsham/senanaly/SenAnaly.htm.

Benhamou, F. and W. J. Older: 1997, 'Applying Interval Arithmetic to Real, Integer, and Boolean Constraints'. *Journal of Logic Programming* **32**(1), 1–24.

Cleary, J.: 1987, 'Logical arithmetic'. *Future Computing Systems* **2**, 125–149.

Davoren, J. and A. Nerode: 2000, 'Logics for Hybrid Systems'. *Proceedings of the IEEE* **88**(7), 985–1010.

de Bakker, J., C. Huizing, W. de Roever, and G. Rozenberg (eds.): 1991, 'Real-Time: Theory in Practice. REX Workshop', Vol. 600 of *Lecture Notes in Computer Science*. Mook, The Netherlands: REX (Research and Education in Concurrent Systems, Springer Verlag.

Delzanno, G. and A. Podelski: 1999, 'Model Checking in CLP'. In: R. Cleaveland (ed.): *Proceedings of the Fifth International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS '99)*, Vol. 1579 of *Lecture Notes in Computer Science*. pp. 223–239.

Delzanno, G. and A. Podelski: 2001, 'Constraint-based Deductive Model Checking'. *International Journal on Software Tools for Technology Transfer (STTT)* **3**(3).

Deransart, P., A. Ed-Dbali, and L. Cervoni: 1996, *Prolog: The Standard; Reference Manual*. Springer Verlag.

Diaz, D.: 2002, 'GNU Prolog Manual'. 1.7 edition.

Fahrland, D. A.: 1970, 'Combined discrete event continuous systems simulation'. *Simulation* **14**(2), 61–72.

Gupta, V., R. Jagadeesan, and V. Saraswat: 1996, '`Hybrid cc`, Hybrid Automata and Program Verification'. In: R. Alur, T. A. Henzinger, and E. D. Sontag (eds.): *Hybrid Systems III: Verification and Control*, Vol. 1066 of *Lecture Notes in Computer Science*. pp. 52–63.

Gupta, V., R. Jagadeesan, V. Saraswat, and D. G. Bobrow: 1995, 'Programming in hybrid constraint languages'. In: P. Antsaklis, W. Kohn, A. Nerode, and S. Sastry (eds.): *Hybrid Systems II*, Vol. 999 of *Lecture Notes in Computer Science*. pp. 226–251.

Henzinger, T. A.: 1996, 'The Theory of Hybrid Automata'. In: *Proceedings, 11th Symposium on Logic in Computer Science (LICS '96)*. pp. 278–292.

Henzinger, T. A., B. Horowitz, R. Majumdar, and H. Wong-Toi: 2000, 'Beyond HyTech: Hybrid Systems Analyis Using Interval Numerical Methods'. In: N. Lynch and B. H. Krogh (eds.): *Hybrid Systems: Computation and Control (HSCC 2000)*, Vol. 1790 of *Lecture Notes in Computer Science*. pp. 130–144.

Hickey, T. J.: 2000, 'CLIP: A CLP(Intervals) Dialect for Metalevel Constraint Solving'. In: E. Pontelli and V. S. Costa (eds.): *Practical Aspects of Declarative Languages: PADL 2000*, Vol. 1753 of *Lecture Notes in Computer Science*. pp. 200–214. A later version is (Hickey, 2001).

Hickey, T. J.: 2001, 'Metalevel Interval Arithmetic and Verifiable Constraint Solving'. *Journal of Functional and Logic Programming* **2001**(7). http://danae.uni-muenster.de/lehre/kuchen/JFLP/articles/2001/S01-02/JFLP-A01-07.pdf.

Hickey, T. J. and Q. Ju, 'clip 1.0 A CLP(Intervals) interpreter, based on Sicstus Prolog'. interval.sourceforge.net/interval/prolog/clip.

Hickey, T. J. and Q. Ju.: 1997, 'Efficient Implementation of Interval Arithmetic Narrowing Using IEEE Arithmetic'. Technical report, Brandeis University CS Dept. www.cs.brandeis.edu/∼tim/narrow_multiply/paper.ps.

Hickey, T. J. and D. K. Wittenberg: 2004, 'Rigorous Modeling of Hybrid Systems using Interval Arithmetic Constraints'. In: R. Alur and G. J. Pappas (eds.): *Hybrid Systems: Computation and Control HSCC 2004*, Vol. 2993 of *Lecture Notes in Computer Science*. pp. 402–416.

Holzbaur, C.: 1995, 'OFAI CLP(Q,R) Manual'. Austrian Research Institute for Artificial Intelligence, Vienna, 1.3.3 edition. TR-95-05.

Jaffar, J. and J. Lassez: 1987, 'Constraint Logic Programming'. In: *Proceedings 14th ACM Symposium on the Principles of Programming Languages*. pp. 111–119.

Jaffar, J. and M. J. Maher: 1994, 'Constraint Logic Programming: A Survey'. *Journal of Logic Programming* **19/20**, 503–581.

Jaffar, J., S. Michaylov, P. J. Stuckey, and R. H. C. Yap: 1992, 'The CLP($\mathcal{R}$) Language and System'. *ACM Transactions on Programing Languages and Systems* **14**(3), 339–395.

Johansson, K. H., M. Egerstedt, J. Lygeros, and S. Sastry: 1999, 'On the regularization of Zeno hybrid automata'. *System and Control Letters* **38**, 141–150.

Kowalewski, S., O. Stursberg, M. Fritz, H. Graf, I. Hoffman, J. Preußig, M. Remelhe, S. Simon, and H. Treseler: 1999, 'A Case Study in Tool-Aided Analysis of Discretely Controlled Continuous Systems: The Two Tanks Problem'. In: P. Antsaklis, W. Kohn, M. Lemmon, A. Nerode, and S. Sastry (eds.): *Hybrid Systems V*, Vol. 1567 of *Lecture Notes in Computer Science*. pp. 163–185.

Lamport, L.: 1986, 'Buridan's Principle'. Revised from a version of Oct. 1984; http://research.microsoft.com/users/lamport/pubs/buridan.ps.

Lynch, N., R. Segala, and F. Vaandrager: 2001, 'Hybrid I/O Automata Revisited'. In: M. D. D. Benedetto and A. Sangiovanni-Vincentelli (eds.): *Hybrid Systems: Communication and Control*, Vol. 2034 of *Lecture Notes in Computer Science*. pp. 403–417.

Lynch, N., R. Segala, F. W. Vaandrager, and H. Weinberg: 1999, 'Hybrid I/O automata'. Technical Report CSI-R9907, Computing Science Institue Nijmegen; Faculty of Mathematics and Informatics; Catholic University of Nijmegen, Toernooivveld 1; 6525 ED Nijmegen; The Netherlands.

Maler, O., Z. Manna, and A. Pnueli: 1991, 'From Timed to Hybrid Systems'. in (de Bakker et al., 1991), pp. 447–484.

Moore, R. E.: 1966, *Interval Analysis*. Prentice-Hall.

Older, W. and A. Vellino: 1993, 'Constraint Arithmetic on Real Intervals'. In: A. Colmerauer and F. Benhamou (eds.): *Constraint Logic Programming: Selected Research*. MIT Press.

Prolog 95: 1995, 'ISO Prolog Standard ISO/IEC 13211-1, Information Technology — Programming Languages — Prolog — Part 1: General Core'. available from www.iso.org/iso/en/.

Research, B. N.: 1988, *BNR Prolog user guide and reference manual*. Bell Northern Research.

sensitivity analysis conference: 2004, 'Fourth International Conference on Sensitivity Analysis of Model Output'. Santa Fe, New Mexico:. http://www.samo2004.org.

Shankar, A. U.: 1993, 'An Introduction to Assertional Reasoning for Concurrent Systems'. *ACM Computing Surveys* **25**(3), 225–262.

Stursberg, O., S. Kowalewski, I. Hoffman, and J. Preußig: 1997, 'Comparing Timed and Hybrid Automata as Approximations of Continuous Systems'. In: P. Antsaklis, W. Kohn, A. Nerode, and S. Sastry (eds.): *Hybrid Systems IV*, Vol. 1273 of *Lecture Notes in Computer Science*. pp. 361–377.

Taylor, J. R.: 1997, *An introduction to Error Analysis: The Study of Uncertainties in Physical Measurements*. University Science Books, second edition.

Urbina, L.: 1996, 'Analysis of Hybrid Systems in CLP($\mathcal{R}$)'. In: E. C. Freuder (ed.): *Principles and Practice of Constraint Programming – CP96*, Vol. 1118 of *Lecture Notes in Computer Science*. pp. 451–467.

Wittenberg, D. K.: 2004, 'CLP(F) Modeling of Hybrid Systems'. Ph.D. thesis, Brandeis University. http://www.cs.brandeis.edu/∼dkw/papers/thesis.ps.

Zhang, J., K. H. Johansson, J. Lygeros, and S. Sastry: 2001, 'Zeno Hybrid Systems'. *International Journal of Robust and Nonlinear Control* **11**(5), 435–451.