Rigorous Modeling of Hybrid Systems using Interval Arithmetic Constraints

Timothy J. Hickey David K. Wittenberg
Technical Report CS-03-241
Computer Science Department
Brandeis University
$\{tim|dkw\}@cs.brandeis.edu$

October 17, 2003

Abstract

We provide a rigorous approach to modeling, simulating, and analyzing Hybrid Systems using CLP(F) (Constraint Logic Programming (Functions))[Hic00], a system which combines CLP (Constraint Language Programming)[JM94] with Interval Arithmetic [Moo66]. We have implemented this system, and provide timing information. Because Hybrid Systems are often used to prove safety properties, it is critical to have a rigorous analysis. By using intervals throughout the system, we make it easier to include measurement errors in our models and safety intervals in our results.

1 Introduction

We wish to rigorously model hybrid systems. Because models of hybrid systems are often used in safety-critical applications, it is crucial to get accurate results with explicit limits on the errors in the model and calculations.

Interval Arithmetic [Moo66][BO97] is an obvious choice for modeling hybrid systems, as the interface between the analog and the digital part involves imperfect hardware whose description must include error bars. Intervals are a very clear way of explicitly expressing error bars. Because Constraint Logic Programming [JM94] provides constraints on the range of values that each variable can take on, it is well suited to proving that certain values are not reached.

We use CLP(F) [Hic00], a CLP language with explicit support for interval arithmetic, to model hybrid systems. Our approach has a formal model (inherited from the formal semantics of CLP) so the results generated by our system can be thought of as theorems about the behavior of the underlying mathematical model of the Hybrid System. These theorems can also be used as lemmas in proofs about safety properties of real Hybrid Systems.

To demonstrate the CLP(F) approach to Hybrid System modeling, we consider the Hybrid System of two interacting water tanks described by Stursberg *et al.* [SKHP97] and later studied by Henzinger*et al.* [HHMWT00]. We provide a one page CLP(F) program for simulating and analyzing the two tanks problem.

2 Related Work

In this paper we describe a practical tool for modeling and analyzing Hybrid Systems. Our tool has a simple formal logical semantics and is based on Interval Arithmetic and Constraint Logic Programming. This paper represents the first attempt to combine these approaches to build a practical tool for rigorously proving properties of Hybrid Systems governed by non-linear ODEs as well as digital logic. In this section we describe the related work on which this paper builds. These foundations are in three separate areas: Hybrid Systems, Constraint Logic Programming, and Interval Arithmetic.

2.1 Formal Models of Hybrid Systems

There has been considerable research on developing formal models of Hybrid Systems which are commonly called Hybrid Automata. Among others, Davoren and Nerode developed logics [DN00], Maler *et al.* [MMP91], Lynch *et al.* [LSVW99] [LSV01], Henzinger *et al.* [Hen96], and Alur *et al.* [ACH+95] 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.

2.2 Practical Modeling/Analysis Tools for Hybrid Systems

Another major research push has been in the development of practical tools for modeling and analyzing realistic Hybrid Systems. For example, the paper of Kowalewski, et.al [KSF+99] describes eight such systems (Matlab, Simulink, gPROMS, Shift, Dymola, BASIP, SMV, HyTech) that can be applied to model and/or analyze fairly complex Hybrid Systems. The cited paper compares these eight systems on a more complex version of the two tanks problem that we consider below. All of these systems sacrifice some degree of rigor in order to handle moderately complex Hybrid Systems. For example, HyTech does not allow the physical system to be governed by ODEs, the other systems that do allow ODEs use some form of approximation that introduces errors which are then ignored. The simulation based systems rely on the user to correctly identify the worst case when there is a non-deterministic choice.

There have been several attempts to modify these techniques to cover non-linear systems. For example, Henzinger *et al.* [HHWT98] translate non-linear systems into linear automata, which they then solve with HyTech [HHWT97]. These translations are a good start, and have been used on real systems, but they add complexity to the software, and are not applicable to all systems.

2.3 CLP approaches to Hybrid Systems

We are not the first to use Constraint Logic Programming to model and analyze Hybrid Systems. Gupta *et al.* [GJSB95][GJS96] introduced a ground breaking approach called "hybrid cc" which allowed one to formally describe Hybrid Systems using a logic programming language with constraints. The resulting models automatically inherit a formal logical semantics from the underlying language thereby allowing program results to be interpreted as formal theorems. The main disadvantage of their approach is that they were restricted to Hybrid Systems with linear ODEs.

Urbina [Urb96] has pioneered another approach using CLP(R) to model and analyze Hybrid Systems. This approach also suffers from a lack of rigor in the handling of the underlying dynamic systems (the CLP(R) system uses floating point arithmetic and does not account for roundoff error, moreover it can only directly solve linear constraints).

Delzanno and Podelski [DP99][DP01] have explored analyzing Hybrid Systems using CLP(Q,R)[Hol95], 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[Sha93] to CLP(Q,R). The system is not able to handle general non-linear ODEs rigorously.

2.4 Interval Arithmetic ODE solving

Floating Point numbers [Kah96] do not have the nice properties (associativity, commutativity ...) that we learned in third grade that all numbers have. This introduces a class of round-off errors in which mathematically equivalent operations give different results. To deal with this (as well as the obvious round-off problems stemming from using a finite set of floating points to model an infinite set of reals) Interval Arithmetic[Moo66] uses an interval (X_{min}, X_{max}) to represent each real number X. The true value of X is within the interval representing X. By careful use of IEEE 754 rounding directives, it is possible to soundly perform arithmetic operations on intervals.

We are not the first to apply Interval Arithmetic techniques to the problem of rigorously modeling Hybrid Systems. Henzinger *et al.* [HHMWT00] take 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. This 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.

HyperTech does standard Hybrid Automata calculations while using intervals in the calculations, rather than as the fundamental unit. This is partially because HyperTech grew out of HyTech[HHWT97] which does all its calculations with infinite precision rationals.

3 Benefits of an Interval Arithmetic Constraints Approach

In this paper we show how the use of CLP(F) [Hic00] allows one to combine many of the best features of prior Hybrid System models.

Our approach can be thought of as combining Henzinger *et al.*'s innovation in using Interval Arithmetic and Urbina's[Urb96] and Gupta *et al.*'s use of constraint systems[GJS96] with the semantic elegance of the Logic Programming approach. In addition, this combination yields some new and potentially useful capabilities, which we highlight in this paper. We use an interval arithmetic ODE constraint solver that automatically and rigorously accounts for the sources of error which other systems often do not handle (choice of case for non-deterministic systems, round-off error, approximation error for the ODEs, ...) and is able to handle highly non-linear ODEs with full mathematical rigor. We use intervals for all calculations and measurements (to model the inevitable error-bars of instruments) as well as to provide over-approximations to deal with rounding error. We avoid the "modeling error" of other IA based ODE approaches by explicitly expressing the ODE as a constraint on function variables. We do this by modeling the system declaratively as a CLP(F) [Hic01] program in which the differential equations appear directly as constraints in the program. The underlying constraint solver allows function variables which may be constrained by non-linear ODEs to generate interval results. The system makes careful use of the IEEE 754[Kah96] floating point rounding directives to obtain provably correct numeric results from standard processors. The resulting CLP(F) program has the property that the results computed using the CLP(F) system are guaranteed to contain all solutions of the ODEs modeled by the constraints. This soundness property is inherited from CLP[JL87][JM94].

One of the major benefits of this approach is that the problem of analyzing the Hybrid System is transformed into the problem of analyzing the corresponding CLP(F) program. In principle, one should be able to apply well understood program analysis techniques to this program and directly infer provable properties of the corresponding Hybrid System. Also the constraint language for specifying ODEs is very expressive and quite close to standard mathematical notation. By making the proof of safety properties more direct, we make it more likely that those proofs will be reliable. In this paper we describe only the simpler types of analysis that one can do by directly solving CLP(F) constraints related to the Hybrid System. A further advantage of our approach is that the system is quite simple, and the programs remarkably simple. We provide the complete program for analyzing the two tanks problem as Figure 1 and Figure 2. The code fits on one page. This makes the argument for the correctness of a result from the system less complex to state. By making the argument for correctness of the system simpler (because the system itself is simpler), we make it less likely that there will be an error in the proof of correctness.

4 CLP(F)

The language CLP(F)[Hic00] is a constraint logic programming language[JL87][JM94] in which the constraints specify arithmetic and analytic relations among real and function variables.

First, we will describe the underlying constraint language of CLP(F), and then discuss the syntax and semantics of CLP(F) programs.

4.1 Analytic Constraints in CLP(F)

In CLP(F) the constraint domain allows one to declare variables representing various analytic values including:

- *real numbers, X*
- *infinitely differentiable functions, F, on a finite interval [a,b]*
- *vectors of numbers, functions, or vectors*

A full description of the language is available in [Hic01] and [Hic00]. In this section, we provide a brief overview of the language, its semantics, its implementation, and its use.

As is common in CLP languages, the constraints are enclosed in curly braces "{}". The different types of variables are declared using the `type` predicate. The CLP(F) interpreter provides answers to queries in the form of a sequence of solution sets, where each solution set provides a real interval for each of the constraint variables. The soundness property of CLP implies that every correct solution to the query must be contained in one of the solution sets (assuming that the program eventually terminates). On the other hand, not every element of the solution set is guaranteed to be a solution (and indeed, there may not be any actual solutions in any particular solution set returned by the interpreter).

4.1.1 Rigorous Numeric Constraint Solving

The CLP(F) constraint language allows one to express any algebraic equality or inequality constraint among real variables. For example,

```
| ?- {X^2=2,X>0}.
X = 1.41421356237309... ? ;
no
| ?-
```

The CLP(F) interpreter represents the interval for X in a compact form. The ellipsis "..." indicates that all shown digits are correct and hence X must lie in the interval:

```
[1.41421356237309,
 1.41421356237310)
```

Also, note that the user entered a semi-colon after the solution and the interpreter responded with "no" which indicates that there are no more solutions.

4.1.2 Multiple Solutions and Non-determinism

Sometimes there maybe more than one solution to a given constraint. The constraint solver will indicate this by returning an interval that contains all solutions:

```
| ?- {X^2=2}.
X = [-1.41421356237309536751922678377,
      1.41421356237309536751922678377] ?
no
| ?-
```

Here, to find the discrete set of solutions one must apply a divide-and-conquer approach where one divides the interval into subintervals and searches for solutions in each one. This is done using the "queue" method of the `solve_clip` solver and typing a semicolon after each solution that it finds:

```
| ?- {X^2=2},solve_clip(queue,[X],0.000001).
X = 1.41421356237309... ? ;
X = -1.41421356237309... ?
(10 ms) no
| ?-
```

The "no" answer at the end, indicates that there are no more solutions to that query.

4.2 Analytic constraints and ODEs

CLP(F) also allows one to constrain functions by functional equations involving the same arithmetic operators and mathematical functions as discussed above. In addition, one can constrain a function to take specific values at specific points and to have a range that lies within an interval.

Consider the following mathematical constraint Q on the function variable F and real variables A and E:

$$Q(F, A, E) \equiv$$
$$(F \in \mathcal{H}([0,1]), F' = F, F([0,1]) \subseteq [-100, 100], F(0) = 1, F(A) = 2, F(1) = E)$$

Q can be represented and solved by presenting the following constraint to the CLP(F) interpreter:

```
| ?- type([F],function(0,1)), {[ ddt(F,1)=F,   F in [-100,100],
       eval(F,0)=1,eval(F,A)=2, eval(F,1)=E ]}.
```

where the `type` predicate indicates that $F \in \mathcal{H}([0,1])$, i.e., F is an analytic function in some open neighborhood of the interval $[0,1]$. The output given by CLP(F) after 0.76 seconds on a 1 GHZ Mac TiBook is

```
A = 0.6931471... E = 2.7182818... ;
(760 ms) no
| ?-
```

which represents the following answer constraint:

$$C(F,A,E) \equiv (A \in [0.6931471, 0.6931472) \ \wedge \ E \in [2.7182818, 2.7182819))$$

The soundness of the CLP(F) interpreter implies that it has proven a theorem about the query and its solution constraint:

$$\forall F, A, E \quad Q(F,A,E) \Rightarrow C(F,A,E)$$

In other words, if F, A, and E represent a solution to Q, then they must satisfy the answer constraint C. Note that one cannot infer from this theorem that Q has any solutions at all. In this particular case, Q clearly does have a solution

$$F(t) = \exp(t), \quad A = \ln(2), \quad E = e$$

which of course satisfies the answer constraint C.

The CLP(F) system solves analytic constraints by soundly approximating analytic functions by power series and introducing arithmetic constraints among the Taylor coefficients of the functions at the endpoints, at points in the interval, and over the entire range. For more details consult [Hic00].

4.3 Programs

CLP(F) programs are Prolog programs in which the bodies of rules may contain CLP(F) constraints. CLP(F) provides the full power of Prolog in addition to the power of the underlying constraint solver and both are combined within a single logical semantics. Moreover, by the soundness and completeness of CLP [JM94] semantics, if a CLP interpreter returns N solutions sets $C_1, \ldots, C_n$ for a query $Q(X,F)$ and then halts, then every solution of the query $Q(X,F)$ consisting of a real vector X and a vector F of real-valued functions, is contained in the union of the solution sets C_i.

The logical semantics of CLP(F) programs can be summarized in the following theorem [Hic00]

Theorem 1 *Let P be a CLP(F) program, $Q(x)$ a CLP(F) query where x is a tuple of real variables, and assume the interpreter returns N answer constraints $\{x \in I_j\}$ for tuples of intervals $I_1, \ldots, I_N$. Let P^* be the first order theory obtained from a logic reading of P (by Clark's Completion Semantics [Cla78][LLo87]), and let T be the first order theory of the domain F of analytic functions on real intervals. Then one can infer that*

$$P^* \cup T \vdash \forall x \ \left(Q(x) \Rightarrow x \in \bigcup_j I_j \right)$$

Theorem 2 *Notation as in the previous theorem. If the interpreter halts with no answer constraints (i.e., N=0), then one can infer*

$$P^* \cup T \vdash \neg \exists x \ Q(x)$$

i.e., the query is not satisfiable.

This theorem allows one to infer correctness of a CLP(F) simulator for a Hybrid System as well as safety properties of the System directly from the corresponding CLP(F) program. We now illustrate this using the Two Tanks example.

5 Two Tanks in CLP

5.1 Description of the Two Tanks Problems

The "Two Tanks" problem is to accurately model a system consisting of two water tanks. There is a flow of water in to the higher tank, and a horizontal pipe from the bottom of the higher tank to some point in the side of the lower tank. There is an outflow pipe at the bottom of the lower tank. In some versions of the problem, there are valves controlling some or all of the input flow, the input to the pipe between the two tanks, and the output flow. The obvious questions to ask are "Is there an equilibrium given a set of flow rates?", "Does either tank overflow before equilibrium is achieved?", and, in the case where the model has valves, "Does some particular program have a specified safety property?"

Kowalewski *et al.*[KSF+99] use 6 methods to model a realistic version of this two tanks problem previously studied by the same group (Stursberg *et al.* [SKHP97]). Later, Henzinger *et al.*[HHMWT00] provided another technique for studying a simplified version of this problem. In this paper, we also study the simplified version with no valves.

Because hybrid automata are often used to model safety properties, it is important that they be correct. This requires a rigorous approach. Any technique to model these systems which does not account for rounding error or error in the approximation to the solution of the ODEs governing the problem, is not rigorous. We use interval arithmetic to provide an explicit limit on rounding errors and on the ODE solution errors.

5.1.1 Mathematics of the Two Tanks Problem

The precise problem we study can be described as follows. There are two tanks, an upper tank and a lower tank. The height of the water in the upper tank at time t is given by $f_1(t)$ and the height in the lower tank is $f_2(t)$. The heights f_1 and f_2 are measured from the bottom of their respective tanks. There is a constant inflow of water into the upper tank (where the flow rate is given by a constant k_1, and a constant flow rate out of the bottom tank given by k_2. The bottom of the upper tank is k_3 meters above the bottom of the lower tank and there is a horizontal pipe connecting the bottom of the upper tank to the lower tank where the flow is governed by the constant k_2. The heights f_1 and f_2 are governed by a pair of ODEs. One pair holds when the water in the lower tank is below the level of the connecting pipe, the other ODE holds when the water level is above the connecting pipe. These ODEs are given below:

$$f_1' = \begin{cases} k_1 - k_2\sqrt{f_1 - f_2 + k_3} & f_2 > k_3 \\ k_1 - k_2\sqrt{(f_1)} & f_2 \leq k_3 \end{cases}$$

$$f_2' = \begin{cases} k_2\sqrt{f_1 - f_2 + k_3} - k_4\sqrt{f_2} & f_2 > k_3 \\ k_2\sqrt{(f_1)} - k_4\sqrt{(f_2)} & f_2 \leq k_3 \end{cases}$$

6 Rigorous Simulation of Hybrid Systems

In this section, we give the complete CLP(F) program describing two tanks problem, as described above, and show how it can be used to rigorously model the behavior of this system.

The program consists of two parts. The first part in Figure 1 describes the relation between the heights of the waters in the two tanks at two times t_0 and t_1. There are four cases considered

- case 1: the lower tank's water level is above the pipe throughout the interval $[t_0, t_1]$
- case 2: the lower tank's water level is below the pipe throughout the interval $[t_0, t_1]$
- case 12: the lower tank's water level is above the pipe at time t_0 and stays above until some point t_2 , at which it is equal to the height of the lower pipe, and then remains below the pipe until time t_1.
- case 21: the symmetric case, where the water level rises from t_0 to t_1 and is equal to the height of the pipe at exactly one time t_2.

The code in the figure is a straightforward representation of these cases. For example, we use the range constraints `X2 in [K2,1000]` to specify that the height of the water in the second tank is always above `K2`. The upper bound for the height of the water is specified to be 1000 for performance reasons. Providing a finite upper bound speeds some calculations greatly. Note that the problem of finding the transition point t_2 is automatically handled by the underlying CLP(F) system by simply adding the constraint `X2a=K3`.

```
twotank(case1,X10,X20,T0,X11,X21,T1,[K1,K2,K3,K4]) :-
        decls([X1,X2],function(T0,T1)),
     {[  ddt(X1,1) = K1 - K2*psqrt(X1-X2+K3),
         ddt(X2,1) = K2*psqrt(X1-X2+K3) - K4*psqrt(X2),
         eval(X1,T0)=X10,   eval(X1,T1)=X11,
         eval(X2,T0)=X20,   eval(X2,T1)=X21,
         X1 in [E,1000],         X2 in [K3,1000],
    ]}.

twotank(case2,X10,X20,T0,X11,X21,T1,[K1,K2,K3,K4]) :-
 decls([X1,X2,Z1,Z2],function(T0,T1)),
      {[ ddt(X1,1) = K1 - K2*psqrt(X1),
         ddt(X2,1) = K2*psqrt(X1) - K4*psqrt(X2),
         eval(X1,T0)=X10,   eval(X1,T1)=X11,
         eval(X2,T0)=X20,   eval(X2,T1)=X21,
         X1 in [E,1000],      X2 in [E,K3],
    ]}.
.
twotank(case12,X10,X20,T0,X11,X21,T1,Ks) :-
   {T0=<Ta, Ta<T1},Ks=[_,_,K3,_],{X2a=K3},
   twotank(case1,X10,X20,T0,X1a,X2a,Ta,Ks),
   nl,nl,print(case12(X1a,X2a,Ta)),nl,nl,
   twotank(case2,X1a,X2a,Ta,X11,X21,T1,Ks).

twotank(case21,X10,X20,T0,X11,X21,T1,Ks) :-
   {T0=<Ta, Ta<T1},   Ks=[_,_,K3,_],{X2a=K3},
   twotank(case2,X10,X20,T0,X1a,X2a,Ta,Ks),
   nl,nl,print(case21(X1a,X2a,Ta)),nl,nl,
   twotank(case1,X1a,X2a,Ta,X11,X21,T1,Ks).

   % equilibrium is at  X10=0.625,  X20=0.5625,
ks([K1,K2,K3,K4]) :-   K2=1, K4=1, % sqrt(meters)/second
                                     K3=0.5, % meters
                                     K1= 0.75 % meters/sec
```

Figure 1: CLP(F) code for Case1, Case2, and transitions between them

```
iterate(N,_DT,X10,X20,T0,X10,X20,T0,_Ks) :-   {N<0},fail.
iterate(N,_DT,X10,X20,T0,X10,X20,T0,_Ks) :-   {N=0}.
iterate(N,DT,X10,X20,T0,X11,X21,T1,Ks) :-   {T1a=T0+DT,   N1=N-1},
 forward_chk([X1a,X2a,T1a], twotank(Case,X10,X20,T0,X1a,X2a,T1a,Ks))
 iterate(N1,DT,X1a,X2a,T1a,X11,X21,T1,Ks).
```

Figure 2: CLP(F) code for iterating to find a fixpoint

The second part of the program is an iterator (Figure 2) that repeatedly steps through the time domain applying the appropriate case (or when nondeterminism is present, cases) to compute the current water levels in the two tanks. This program makes the assumption that the water level does not cross the height of the pipe more than once in any DT interval. We could handle this by making the program a little more complex, but for presentation purposes we stick to this simple case for now. (We would need to use an adaptive step size when switching from case 1 to case 2 or back). Also, the `forward_chk` call is used as a space optimization to minimize the size of the constraint set.

This program can now be executed by loading it into the CLP(F) interpreter and posing queries. For example, in Figure 3 we show the (slightly edited) output results of a query that rigorously follows the water levels over a period of two seconds with 0.1 second steps. Note that it finds the transition point from case 2 to case 1 automatically.

7 Rigorous Analysis of Hybrid Systems

The same program can be used to prove properties of the two tanks system. In this section we show how to prove the following safety property, which states that if the tank levels are ever sufficiently close to an "equilibrium" point, then they stay relatively near that point forever, more precisely

If the tank levels X_0 for the upper tank and Y_0 for the lower tank satisfy

$$0.62 \leq X_0 \leq 0.63 \quad \wedge \quad 0.558 \leq Y_0 \leq 0.567$$

at time 0, then for all times t in the future the tank levels X and Y satisfy

$$0.61922 \leq X \leq 0.63083 \wedge 0.55674 \leq Y \leq 0.56815$$

We prove this in two steps. First we prove that if the tank levels are in the initial interval $[0.62, 0.63] \times [0.558, 0.567]$ at time 0, then they are also in that interval at time 0.1. This implies that they are in that interval at time $N * 0.1$ for all N. Next we prove that if they start in the given interval at time 0, then they are in the second stated interval at all times t with $0 \leq t \leq 0.1$. This proves the safety property.

The first part can be proved directly by using the "solve_clip" solver which provides increasingly more precise bounds on the answer constraint as shown in Figure 4. This corresponds to the standard Interval Arithmetic ODE solving approach. In our system, it takes about 10 minutes to prove this directly.

Another approach is to use constraints and try to find an initial point (X, Y) such that after 0.1 seconds it is "out of the box". This is specified by the query in Figure 5. As can be seen, this returns with a "no" answer, which means no such point exists and hence all such (X, Y) must end up inside the "box". The calculation takes about 3 seconds and is more elegant than the direct approach

The last part of the proof, involves computing the range of possible values of (X, Y) over the interval $[0, 0.1]$ assuming they start in the specified box. This is done by making the query in Figure 6. in about 2.3 seconds.

8 Advantages

There are several advantages to our approach.

```
| ?- reset_clip, ks(Ks),iterate(N,0.1,0.75,0.375,0,  X,Y,T, Ks).

N = 0  X = 0.75 Y = 0.375 ? ;
N = 1 X = 7.3872686208537...e-01 Y = 3.9904710750...e-01 ? ;
N = 2 X = 7.280907797217...e-01  Y = 4.2065058557...e-01 ? ;
N = 3 X = 7.180600004968...e-01  Y = 4.4006784395...e-01 ? ;
N = 4 X = 7.086039345668...e-01  Y = 4.575212542...e-01 ? ;
N = 5 X = 6.9969315162...e-01    Y = 4.732050513...e-01 ? ;
N = 6 X = 6.9129937388...e-01    Y = 4.87290407...e-01 ? ;
N = 7 X = 6.833954653...e-01     Y = 4.9992926...e-01 ? ;

case21(6.8335011...e-01,REAL([4.9999*e-01, 5.0000*e-01]),7.005915...e-01)

N = 8 X = 6.762886...e-01 Y = 5.109318...e-01 ? ;
N = 9 X = 6.702047...e-01 Y = 5.202036...e-01 ? ;
N = 10 X = 6.64998...e-01 Y = 5.280075...e-01 ? ;
N = 11 X = 6.60542...e-01 Y = 5.34567...e-01 ? ;
N = 12 X = 6.5672...e-01  Y = 5.4007...e-01 ? ;
N = 13 X = 6.534...e-01   Y = 5.4469...e-01 ? ;
N = 14 X = 6.506...e-01   Y = 5.485...e-01 ? ;
N = 15 X = 6.482...e-01   Y = 5.51...e-01 ? ;
N = 16 X = 6.461...e-01   Y = 5.54...e-01 ? ;
N = 17 X = 6.44...e-01    Y = 5.56...e-01 ? ;
N = 18 X = [6.42685e-01, 6.43255e-01]  Y = 5.58...e-01 ? ;
N = 19 X = [6.26485e-01, 6.69285e-01]  Y = [5.57665e-01, 5.62725e-01] ? ;
N = 20 X = [6.08805e-01, 6.88585e-01]) Y = [5.41135e-01, 5.78425e-01] ?
| ?-
```

Figure 3: CLP(F) results showing transition between cases

```
| ?-  {X0 = [0.62,0.63],Y0=[0.558,0.567]},
        ks(Ks), twotank(case1,X0,Y0,0.0,X,Y,0.1,Ks)
get_range(X,Y),solve_clip(fwchk,[X,Y],N),get_bounds(X,Lx,Hx),
        get_bounds(Y,Ly,Hy).

N = 0
X = [.619185, .630855]   Y = [.556695, .568205] ? ;

N = 1  X = [.619615, .630375]  Y = [.557575, .567425] ? ;
N = 2  X = [.619835, .630155]  Y = [.557955, .567045] ? ;
N = 3  X = [.619945, .630045]  Y = [.558115, .566875] ? ;
N = 4  X = [.619985, .629995]  Y = [.558185, .566805] ? ;
N = 5  X = [.620015, .629975]  Y = [.558215, .566775] ?

(633920 ms) yes
| ?-
```

Figure 4: IA direct proof of safety property

```
| ?-  {X10 = [0.62,0.63],X20=[0.558,0.567]},
    ks(Ks), twotank(case1,X10,X20,0.0,X11,X21,0.1,Ks),
   ({X11<0.62} ; {X11>0.63}; {X21<0.558}; {X21>0.567}).

(2890 ms) no
| ?-
```

Figure 5: Safety Property Proof via negative answer

```
 ?- {T = [0,0.1]}, {X10 = [0.62,0.63],X20=[0.558,0.567]},
    ks(Ks), twotank(case1,X10,X20,0.0,X,Y,T,Ks).

T = REAL(0,0.10000000000000001942890293094)
X = [0.61922,0.63083]
Y= [0.56574,0.56815]

(2350 ms) yes
```

Figure 6: computation of range over $[0,1]$ in Safety analysis

- The paradigm allows simpler proofs. We can prove convergence by splitting a region into areas, and showing that each of those areas eventually leads to a loop.
- The system can rigorously handle non-linear ODEs.
- The semantics of CLP are closer to the ODEs describing the problem. The problem specification is translated trivially into a program.
- By making the argument for correctness of the system simpler (because the system itself is simpler), we make it less likely that there will be an error in the proof of correctness.
- While CLIP is limited to analytic functions, it can handle points at which a function is not analytic, as long as the function is continuous (or nearly so) at all points. One simply writes one function for values above the non-analytic point and another for values below.

9 Limitations

The primary disadvantages of this approach are that it is very resource intensive and hence can not currently model systems over a long modeling period and that it does not yet handle the wrapping problem for the Interval Arithmetic calculation of ODEs. The wrapping problem [MA85] is that in multi-dimensional interval arithmetic, the interval is always rectangular (a hyper-cube). This rectangle is often much larger than the minimum volume shape to cover all possible values. This excessive over-approximation can make true statements unprovable. The simple minded solution to the wrapping problem is to divide each rectangle into smaller pieces, exacerbating the performance problems.

References

[ACH+95] Rajeev Alur, Costas Courcoubetis, Nicolas Halbwachs, Thomas A. Henzinger, Pei-Hsin Ho, Xavier Nicollin, Alfredo Olivero, Joseph Sifakis, and Sergio Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138:3–34, 1995.

[BO97] Frédéric Benhamou and William J. Older. Applying interval arithmetic to real, integer, and boolean constraints. *Journal of Logic Programming*, 32(1):1–24, Jul 1997.

[Cla78] Keith Clark. Negation as failure. In Herve Gallaire and Jack Minker, editors, *Logic and Databases*, pages 293–322. Plenum Press, New York, NY, 1978.

[DN00] J.M. Davoren and Anil Nerode. Logics for hybrid systems. *Proceedings of the IEEE*, 88(7):985–1010, Jul 2000.

[DP99] Giorgio Delzanno and Andreas Podelski. Model checking in CLP. In Rance Cleaveland, editor, *Proceedings of the Fifth International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS '99)*, volume 1579 of *LNCS*, pages 223–239. Springer Verlag, 1999.

[DP01] Giorgio Delzanno and Andreas Podelski. Constraint-based deductive model checking. *International Journal on Software Tools for Technology Transfer (STTT)*, 3(3), 2001.

[GJS96] Vineet Gupta, Radha Jagadeesan, and Vijay Saraswat. `Hybrid cc`, hybrid automata and program verification. In Rajeev Alur, Thomas A. Henzinger, and Eduardo D. Sontag, editors, *Hybrid Systems III: Verification and Control*, volume 1066 of *LNCS*, pages 52–63. Springer Verlag, 1996.

[GJSB95] Vineet Gupta, Radha Jagadeesan, Vijay Saraswat, and Daniel G. Bobrow. Programming in hybrid constraint languages. In Panos Antsaklis, Wolf Kohn, Anil Nerode, and Shankar Sastry, editors, *Hybrid Systems II*, volume 999 of *LNCS*, pages 226–251. Springer Verlag, 1995.

[Hen96] Thomas A. Henzinger. The theory of hybrid automata. In *Proceedings, 11th Symposium on Logic in Computer Science (LICS '96)*, pages 278–292. IEEE Computer Society Press, 1996.

[HHMWT00] Thomas A. Henzinger, Benjamin Horowitz, Rupak Majumdar, and Howard Wong-Toi. Beyond HyTech: Hybrid systems analyis using interval numerical methods. In Nancy Lynch and Bruce H. Krogh, editors, *Hybrid Systems: Computation and Control (HSCC 2000)*, volume 1790 of *LNCS*, pages 130–144. Springer Verlag, 2000.

[HHWT97] Thomas A. Henzinger, Pei-Hsin Ho, and Howard Wong-Toi. HyTech: A model checker for hybrid systems. *Software Tools for Technology Transfer*, 1(?):110–122, 1997.

[HHWT98] Thomas A. Henzinger, Pei-Hsin Ho, and Howard Wong-Toi. Algorithmic analysis of nonlinear hybrid systems. *IEEE Transactions on Automatic Control*, 43:540–554, 1998.

[Hic00] Timothy J. Hickey. Analytic constraint solving and interval arithmetic. In *POPL'00 ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 338–351, 2000. published as vol. 27 of SIGPLAN notices.

[Hic01] Timothy J. Hickey. Metalevel interval arithmetic and verifiable constraint solving. *Journal of Functional and Logic Programming*, 2001(7), October 2001. http://danae.uni-muenster.de/lehre/kuchen/JFLP/articles/2001/S01-02/JFLP-A01-07.pdf.

[Hol95] C. Holzbaur. *OFAI clp(q,r) Manual*. Austrian Research Institute for Artificial Intelligence, Vienna, 1.3.3 edition, 1995. TR-95-05.

[JL87] J. Jaffar and J.L. Lassez. Constraint logic programming. In *Proceedings 14th ACM Symposium on the Principles of Programming Languages*, pages 111–119, 1987.

[JM94] Joxan Jaffar and Michael J. Maher. Constraint logic programming: A survey. *Journal of Logic Programming*, 19/20:503–581, 1994.

[Kah96] W. Kahan. Lecture notes on the status of ieee standard 754 for binary floating-point arithmetic. Technical report, EECS, University of California, Berkely, 1996.

[KSF+99] S. Kowalewski, O. Stursberg, M. Fritz, H. Graf, I. Hoffman, J. Preußig, M. Remelhe, S. Simon, and H. Treseler. A case study in tool-aided analysis of discretely controlled continuous systems: The two tanks problem. In Panos Antsaklis, Wolf Kohn, Michael Lemmon, Anil Nerode, and Shankar Sastry, editors, *Hybrid Systems V*, volume 1567 of *LNCS*, pages 163–185. Springer Verlag, 1999.

[LLo87] John Wylie LLoyd. *Foundations of Logic Programming*. Springer, second, expanded edition, 1987.

[LSV01] Nancy Lynch, Roberto Segala, and Frits Vaandrager. Hybrid I/O automata revisited. In Maria Domenica Di Benedetto and Alberto Sangiovanni-Vincentelli, editors, *HSCC*, number 2034 in LNCS, pages 403–417. Springer Verlag, 2001.

[LSVW99] Nancy Lynch, Roberto Segala, Frits W. Vaandrager, and H.B. Weinberg. 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, Apr 1999.

[MA85] S. Markov and R. Angelov. An interval method for systems of ODE. In Karl Nickel, editor, *Interval Mathematics 1985*, volume 212 of *LNCS*, pages 103–108. Springer, 1985.

[MMP91] Oded Maler, Zohar Manna, and Amir Pnueli. From timed to hybrid systems. In J.W. de Bakker, C. Huizing, W.P. de Roever, and G. Rozenberg, editors, *Real-Time: Theory in Practice*, volume 600 of *LNCS*, pages 447–484. Rex Workshop, Springer Verlag, 1991.

[Moo66] Ramon E. Moore. *Interval Analysis*. Prentice-Hall, 1966.

[Sha93] A. Udaya Shankar. An introduction to assertional reasoning for concurrent systems. *ACM Computing Surveys*, 25(3):225–262, Sep 1993.

[SKHP97] Olaf Stursberg, Stefan Kowalewski, Ingo Hoffman, and Jorg Preußig. Comparing timed and hybrid automata as approximations of continuous systems. In Panos Antsaklis, Wolf Kohn, Anil Nerode, and Shankar Sastry, editors, *Hybrid Systems IV*, volume 1273 of *LNCS*, pages 361–377. Springer Verlag, 1997.

[Urb96] Luis Urbina. Analysis of hybrid systems in CLP($\mathcal{R}$). In Eugene C. Freuder, editor, *Principles and Practice of Constraint Programming – CP96*, volume 1118 of *LNCS*, pages 451–467. Springer, Aug 1996.