Using Analytic CLP to Model and Analyze Hybrid Systems

Timothy J. Hickey and **David K. Wittenberg**
Computer Science Department Brandeis University
Waltham, MA 02454

Abstract

We use CLP(F), an Analytic Constraint Logic Programming (ACLP) language, to model hybrid systems. ACLP languages combine intervals, constraints, and ODEs (Ordinary Differential Equations) in a clean and natural way. CLP(F) provides an implementation of an ACLP language based on interval arithmetic. The semantics of CLP(F) rigorously handle non-linear ODEs and round-off error. The ODEs describing a hybrid system need only a minor change of syntax to become a CLP(F) program. This simple transformation from a physical description of a hybrid system to a program which can be used to provide a proof of safety properties of the system bridges the gap between practical tools and formal models, and allows one to easily prove statements about real-world systems. The combination of interval arithmetic with ACLP makes it easy to pose and answer many sorts of queries about a system. For example, "At what point does the system change from one state to another?", or "What control settings result in a cycle with period t?"

Introduction

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). All computer controlled or monitored processes in the real world are hybrid systems. Recent work on hybrid systems includes defining models (Lynch *et al.* 1999), (Lynch, Segala, & Vaandrager 2001), (Alur *et al.* 1995) (Maler, Manna, & Pnueli 1991), (Gupta, Jagadeesan, & Saraswat 1996) and calculating the behaviour of the analog parts (Henzinger *et al.* 2000). The Intelligent Highway Group at Berkeley has developed the SHIFT programming language (Deshpande, Göllü, & Semenzato) for describing evolving hybrid systems. (Mosterman 1999) provides a survey of a dozen simulation packages describing how much support each of them provides for simulating hybrid systems.

Interval arithmetic (Moore 1966) 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. CLP (Constraint Logic Programming) was introduced by (Jaffar & Lassez 1987). (Jaffar & Maher 1994) provide an excellent survey of CLP. Interval arithmetic fits naturally with CLP, as an interval describing a real value x corresponds to two constraints on x, one for the upper bound, and one for the lower bound. (Benhamou & Older 1997) first combined intervals with CLP. (Bohlender 1996) provides an excellent survey of the literature on "Enclosure Methods" or arithmetic constraints.

We use CLP(F) (Hickey 2000), an analytic CLP language which combines CLP with interval arithmetic. CLP(F) uses interval arithmetic techniques to rigorously solve constraints involving both real and function variables, constrained via arithmetic and functional equations. CLP(F) is related to QSIM (Kuipers 1993) in that each attempts to find an over approximation of the possible states of a system of ODEs (Ordinary Differential Equations), which QSIM calls "qualitative behaviours".

CLP(F) is a particularly good fit for modeling hybrid systems described by ODEs because it handles round-off errors, approximation errors, and measurement errors in a consistent natural way. These are the primary sources of computational difficulties in modeling ODEs. Because interval arithmetic 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 (*ie.*, safety properties).

(Deville, Janssen, & van Hentenryck 2002) explore a technique for minimizing the size of intervals resulting from solving ODEs using constraints and intervals. As they point out, their techniques would fit well with CLP(F), and might improve the performance of CLP(F). Some of the "consistency techniques" they propose are already available in CLP(F) through the `solve_clip` command.

Advantages Over Previous Models of Hybrid Systems

There has been much work in using CLP (Constraint Logic Programming) to analyze various aspects of hybrid systems (Ciarlini & Frühwirth 2000), (Urbina 1996), (Podelski 2000), (Gupta, Jagadeesan, & Saraswat 1996) . One problem with these conventional CLP approaches to modeling hybrid systems is that they must deal with the ODEs describing the continuous part of the system using some sort of approximation, such as discretization into difference equations or restriction to ODEs that have a closed form solution. This introduces a "modeling error" so that the systems are not computationally sound. One must then reason about the modeling error outside of the CLP program. Many systems ignore these errors, and leave it up to the user to understand the numerical instabilities. For example, the SHIFT language (Deshpande, Göllü, & Semenzato) is very expressive, but it solves non-linear ODEs by using a fourth order Runge-Kutta algorithm without bounding the error term, and hence is not rigorous. This sort of numerical analysis (Acton 1996) is very tricky. To require users to understand numerical analysis under pain of getting a wrong answer is to invite error.

(Hickey 2000) describes CLP(F), an Analytic Constraint Logic Programming (ACLP) language over the domain of differentiable functions. In this paper, we show how CLP(F) allows one to overcome this "modeling error" by allowing an ODE to be expressed explicitly as a constraint on function variables. The resulting ACLP 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.

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 (Smith & Hickey 1990) to CLP(F) and directly infer provable properties of the corresponding hybrid system. 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. The primary disadvantage of this approach is that it is very resource intensive and hence can not currently model systems over a long modeling period.

To demonstrate the ACLP approach to hybrid system modeling, we consider the hybrid system of a thermostat introduced in (Henzinger, Ho, & Wong-Toi 1998). This is a system consisting of a stirred pot of water with a temperature sensor and a heater in it. When the measured temperature goes above a threshold, the logic circuit shuts off the heater (after a small delay). Similarly, when the measured temperature goes below a threshold, the logic circuit turns on the heater (after a small delay). The safety property in question is to establish upper and lower bounds for the temperature of the water. The state diagram is given in Figure 2.

Henzinger *et al.* take a major step towards reliability of their results by using interval arithmetic in solving the differential equations which describe the system. We improve on this by modeling the system declaratively as an ACLP program (written in CLP(F) (Hickey 2001)) in which the differential equations appear directly as constraints in the program and the system is modeled using intervals for all measurements (to model the inevitable error-bars of instruments) as well as to provide over-approximations to deal with rounding error.

CLP(F)

CLP(F) allows one to constrain functions by functional equations involving standard arithmetic operations, trigonometric functions, and exponential functions. In addition, one can constrain a function to take certain values at certain points and to have a range that lies within an interval.

The CLP(F) system solves analytic constraints by soundly approximating analytic functions by power series. It can then introduce arithmetic constraints among the Taylor coefficients of the functions at the endpoints, at points in the interval, and over the entire range. Since CLP(F) represents functions as Taylor series, it can easily calculate derivatives of functions, and enforce constraints on those derivatives. The CLP(F) solver can handle very complex non-linear differential equations as it is based on a "brute force" reduction of the analytic constraints into arithmetic constraints which are solved with a simple interval arithmetic constraint solver.

For example, the following constraint specifies that F is a function on $[0,1]$ such that $F' = F$ and $F(0) = 1$ and $F(A) = 2$ and $F(1) = E$ and $F([0,1]) \subset [-1000, 1000]$:

```
| ?- type([F],function(0,1)),
      {[ ddt(F,1)=F, eval(F,0)=1,
         eval(F,A)=2, eval(F,1)=E,
         F in [-1000,1000] ]}.
A = 0.6931471... E = 2.7182818...
(760 ms) no
```

The `type` predicate is used to declare that F is an infinitely differentiable function on the interval $[0,1]$. Thus F is represented by a list of its Taylor coefficients at 0 ($F_{00}, F_{01}, F_{02}, .., F_{0n}$) and at 1 ($F_{10}, F_{11}, ...$) and the ranges of its derivatives over [0,1] ($R_0, R_1, ...$), related by the Taylor formula with remainder. The function F is then constrained to be equal to its first derivative (i.e. $F_{ij} = F_{i,j+1}, R_i = R_{i+1}$, and to take the value 1 at 0 ($F_{00} = 1$) and to take values in $[-1000, 1000]$ for all $x \in [0,1]$ (i.e. $R_0 \subseteq [-1000, 1000]$). The variables A and E are not declared to be functions and hence are real constants by default. They are constrained so that $F(A) = 2$ and $F(1) = E$ (e.g. $F_{10} = E$ and for each $n = 1, 2, \ldots$, $2 = F_{00} + F_{01}A + F_{02}A^2/2! + \ldots + Z_n A^n/n!$ for some $Z_n \in R_n$). The constraint solver finds A and E to 7 decimal digits of precision and also finds an interval for F (not shown here) and specifies intervals F_{ij} for its first 10 derivatives at 0 and 1, and intervals R_j for the range of its first 10 derivatives over $[0,1]$.

In this paper we will use CLP(F) to define higher order constraints which specify that two points lie on a trajectory defined by on ODE. For example, in the simplest model of a thermostat we use the following CLP(F) procedure, where `T0,T1` are times and `A0,A1` are temperatures at those times, A is the temperature function (so $A(T0) = A0$) and `Alpha,Beta` are the heat loss and the heater element components of the ODE for A. The parameter I is a bound on the width of the interval on which A is defined. This bound is required because in CLP(F) all functions must be defined on finite intervals.

```
ode((T0,A0),[I,[Alpha,Beta]],
                  A,(T1,A1)) :-
   type([A],function(0,I)),
{[ ddt(A,1) = Alpha * A   +   Beta,
    eval(A,0)=A0,   eval(A,T)=A1,
    A in [-1.0E100,1.0E100],
```

```
    T=T1-T0,   T in [0,I]   ]}.
```

The CLP(F) system is easily able to use this definition to compute $(T1, A1)$ from $(T0, A0)$, or, as we will see below, to use this procedure to find values of the parameters `Alpha` and `Beta` which make the system behave in some desired fashion.

In this case the ODE is $f' = af + b, f(0) = a_0, f(t) = a_1$ which can be solved exactly. CLP(F) can handle trigonometric or exponential functions as well as the linear functions shown here, but we restrict ourselves to linear functions in this paper due to space restrictions. Since CLP(F) uses brute force to model ODEs, it does not perform better on ODEs which are solvable analytically. See (Hickey & Wittenberg 2003) for examples of CLP(F) working on more complex functions.

Programs

The program in Figure 1 is one way of implementing a general hybrid system simulator in CLP(F). The first parameter of the `evolve` procedure is the initial state of the hybrid system, which consists of a discrete state S and a continuous state X. The second parameter is a list of values used to specify the particular hybrid system. The third parameter is the final (or ending) state of the hybrid system.

```
evolve(H,C,H,[]).

evolve((S0,X0),C,(S,X)) :-
   statechange((S0,X0),C,S1),
   in_trajectory((S1,X0),C,X1),
   evolve((S1,X1),C,(S,X)).
```

Figure 1: A general simulator for hybrid systems

Observe that the procedure is simple. It looks for a state change from `(S0,X0)` to state `S1`. Such a change may constrain the values of `X0` to lie within narrow intervals. Then it looks for a new trajectory represented by the continuous variable `X1`. Typically, the continuous state will be represented by a pair of dependent variables `(T,A)` where A is the value of some ideal system sensor at time T. The `in_trajectory` procedure looks up the ODE, C, that should hold in this state and applies that ODE to the initial values `X0` to get the new values `X1`.

Henzinger's Model and Analysis

In this section, we present the model of a thermostat with a delay in switching used by (Henzinger *et al.* 2000). Henzinger's model consists of a finite state controller with an analog input measuring the temperature in the tank. The controller has a 1-bit output to control a heater in the tank. The tank always loses heat at a rate directly proportional to the temperature, and, while the heater is on, is heated at 4 degrees/second. Mathematically, after the heater reaches equilibrium in the on position $A' = -A + 4$ and at equilibrium in the off position $A' = -A$. The controller switches the heater off within one second of the temperature going above a pre-set value, and turns the heater on within one second of the temperature dropping below another threshold. Note that this model assumes that the thermometer is perfect, the heater produces a constant and perfectly known heat output, the element heats and cools instantly and the physics of the tank are perfectly modeled by the ODE. Given those assumptions, they then use interval techniques to eliminate round off errors in proving safety properties.

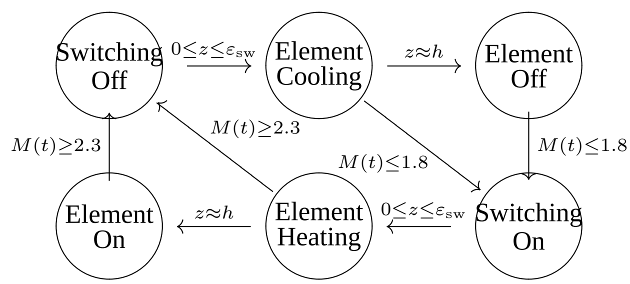

Figure 2: State model allowing thermostat to shut off before element is warm

CLP(F) Model of the Thermostat

In this section we present two models of a thermostat. The first simple model demonstrates the key ideas. The second illustrates how one can easily extend the simple model to a model that more faithfully represents the real hybrid system by more closely approximating the physics of the system.

The Simple Model

Our first CLP(F) model of a thermostat is shown in Figure 3. To clarify the key concepts, this first model assumes there are only two states: on and off. When the system state is on, the ODE governing the temperature A is $A' = -A + 4$. When the system state is off, the ODE is $A' = -A$. The system switches from on to off when the temperature rises above 2.3 and it switches from off to on when it drops below 1.8. The `in_trajectory` procedure models the trajectory by looking up the proper ODE for the current state and then calling the ODE procedure to constrain the new state variables `(T1,A1)`. It also adds the constraint that the temperature range is contained in `[-1000, 2.3]` (resp. `[1.8,1000]`).

This is not needed for our simple example because the temperature rises monotonically and then falls monotonically and then rises again. With more complex models, the temperature might not behave so nicely so this constraint states that no point in the trajectory has passed the threshold for switching. The `statechange` procedure simply indicates the condition that signals a state change and provides the new state. The `ode` procedure models the specified ODE as we have described above. Finally the `test` procedure shows how this program can be used to model the behavior of the system. It initializes the list describing the system to be analyzed and then invokes the `evolve` procedure.

A simple query ("at what times is the temperature 2?") to this system and the resulting answer is shown in Figure 4.

```
evolve((S0,T0,A0),C,(S1,T1,A1)) :-
   S0=S1,{[T0=T1,A0=A1]}.
evolve((S0,X0),C,(S,X)) :-
   statechange((S0,X0),C,S1),
   in_trajectory((S1,X0),C,X1),
   evolve((S1,X1),C,(S,X)).

in_trajectory((S0,(T0,A0)),
        [I, Min, Max, ODEs],(T1,A1)) :-
 member(S0=ODE,ODEs), {T=T1-T0,T=<I},
 ode((T0,A0),[T,ODE],A,(T1,A1)),
 ( (S0=on,      {[A in [-1000,Max] ]});
   (S0=off,     {[A in [Min,1000]]})).

statechange((S0,(T0,A0)),
            [_I, Min, Max, ODEs],S1) :-
 ( (S0=on,      {A0= Max},  S1=off);
   (S0=off      {A0=Min },  S1=on) ).

ode((T0,A0),[I,[Alpha,Beta]]
                        ,A,(T1,A1)) :-
   type([A],function(0,I)),
{[ ddt(A,1) = Alpha * A + Beta,
    eval(A,0)=A0,   eval(A,T)=A1,
    A in [-1.0E100,1.0E100],
    T=T1-T0,   T in [0,I]]}.

test(S,X) :-
 C=[ 2.0,   1.8, 2.3,
 [on=[-1,4],off=[-1,0]]],
 in_trajectory((on,(0,2)),C,X0),
 evolve((on,X0),C,(S,X)).
```

Figure 3: Simplest CLP(F) model of a thermostat

One subtle point about this model is that the CLP(F) solver will only work effectively if a finite step size is given explicitly (this is the `I` parameter appearing in the `in_trajectory` and `ode` procedures. If the step size is too large, then the CLP(F) solver will return very wide, unhelpful intervals for all variables.

One approach to handling this is to introduce pseudo states `(on,n)`, `(off,n)`, where n is an integer representing the number of full steps that have been taken on the current trajectory in the current state. The continuous part can be modeled as `(t,a,z)` where t is the total elapsed time, a is the temperature at time t, and z is the time relative to the current step. Such an extension of the current technique is straightforward and we do not show it here due to space limitations.

```
| ?- test(S,(T,A)),{A=2}.
A = 2, S = on,  T = 0 ?
A = 2, S = off, T = 0.3022808718... ? ;
A = 2, S = on,  T = 0.5029515673... ?
```

Figure 4: Query to Simple Model - When is Temp = 2?

```
evolve((S,T,A,Z),_,(S1,T1,A1,Z1)) :-
   S=S1,{[T=T1,A=A1,Z=Z1]}.
evolve((S0,X0),C,(S,X)) :-
   statechange((S0,X0),C,S1),
   in_trajectory((S1,X0),C,X1),
   evolve((S1,X1),C,(S,X)).

in_trajectory((S0,(T0,A0,_Z0)),
 [Step, Min, Max, Delay, Stime, ODEs],
                  (T1,A1,Z1)) :-
 member(S0=ODE,ODEs),
 {Z1=T, T=T1-T0, T=<Step},
 ode((T0,A0),[T,ODE],A,(T1,A1)),
 ((S0=on,      {[A in [-1000,Max] ]});
  (S0=sw0,     {[T=<Delay]});
  (S0=cooling, {[T<Stime,
                 A in [Min,1000]]});
  (S0=off,     {[A in [Min,1000] ]});
  (S0=sw1,     {[T=<Delay]});
  (S0=heating, {[T<Stime,
                 A in [-1000,Max]]})).

statechange((S0,(_T0,A0,T)),
 [_S, Min, Max, Delay,Stime,_O],S1) :-
 ((S0=on,     {A0=Max},   S1 = sw0);
  (S0=sw0,    {T=Delay},  S1=cooling);
  (S0=cooling,{T=Stime},  S1=off);
  (S0=cooling,{A0=Min},   S1=sw1);
  (S0=off,    {A0=Min },  S1=sw1);
  (S0=sw1,    {T=Delay},  S1=heating);
  (S0=heating,{T=Stime},  S1= on);
  (S0=heating,{A0=Max},   S1= sw0)).

ode((T0,A0),
    [I,[Alpha,Beta,Gamma,Delta]],
  A,(T1,A1)) :-
   type([A,B],function(0,I)),
{[  ddt(A,1) = Alpha*A +Beta +Gamma*B,
    ddt(B,1) = Delta*B,
    eval(A,0)=A0,   eval(A,T)=A1,
    eval(B,0)=1,
    A in [-1.0E100,1.0E100],
    B in [-1.0E100,1.0E100],
    T=T1-T0,   T in [0,I]
]}.

test(S,X,D) :-
  C=[ 2.0,   1.8, 2.3, 0.05, 0.1,
     [on=[-1,4, 0,1],off=[-1,0,0,1],
     sw0=[-1,4, 0,1],sw1=[-1,0,0,1],
 heating=[-1,4,-4,D],
 cooling=[-1,0, 4,D]  ]],
  in_trajectory((on,(0,2,0)),C,X0),
  evolve((on,X0),C,(S,X)).
```

Figure 5: More Complete Model of Thermostat

A More Realistic Model

In the example shown in Figure 5, we refine the previous model by using six states `on,sw0,cooling,off,sw1,heating` corresponding to the states in Henzinger's model. The model also represents the continuous state as a triple T, A, Z where T is the total elapsed time, A is the temperature at time T, and Z is the time since the system entered the current state. The Z parameter is needed to implement the "switching" specification which states that the system waits some amount of time after the threshold is passed before switching on/off the heating element. Likewise, the time in which the system is heating/cooling before it "jumps" to the maximum/minimum value is given by a time unit. This represents a discontinuity in the model since the heating temperature is assumed to immediately rise to the maximum at the end of the element-heating period.

The `sw0,sw1` states represent times when the system is waiting before switching the heating element on or off. The `heating,cooling` states represent times when the element is warming up or cooling down. The `on,off` states represent times when the element is fully on or off. Observe that the ODEs for each state are specified in the variable `C` of the test procedure. Also, observe that the switching conditions are given declaratively in the `statechange` procedure. Finally, note that the system is assumed to be modeled by the following more complex non-linear family of ODEs, where the parameters $(\alpha, \beta, \gamma, \delta)$ vary from state to state:

$$\begin{aligned}
\forall t \in [0,I] \;\; A'(t) &= \alpha A(t) + \beta + \gamma B(t) \\
\forall t \in [0,I] \;\; B'(t) &= \delta B(t) \\
T &= T1 - T0, \;\; 0 \le T \le I \\
A(0) &= A0, \;\; A(T) = A1, \;\; B(0) = 1 \\
A([0,I]), B([0,I]) &\subset [-10^{100}, 10^{100}]
\end{aligned}$$

The variable B represents the heat transfer from the heating element and the rate at which it heats and cools depends on its temperature and on the parameter δ.

The following code shows a more interesting example in which the model is used to find all values of the ODE parameter δ in the range $[-10, -5]$ for which the system evolves to the state with `S=off` and $A = 2$ in exactly 0.5 seconds.

```
| ?- {D in [-10,-5],T=1/2,A=2}, S=off,
test(S,(T,A,Z),D),narrow_all(10000000).

A = 2
D = -8.3533433047...e+00
S = off
T = 0.5
Z = 1.87481070502225...e-01 ?
(10820 ms) yes
```

Conclusions

Strengths of CLP(F)

A novel aspect of the ACLP approach to hybrid system analysis is that it establishes a close correspondence between the semantics of a particular class of constraint programs and the behavior of hybrid systems providing several advantages: The simple mapping means that one needn't worry about the "translation" from ODEs to CLP(F). Interval techniques guarantee that calculated safety properties are correct, and protect against round-off errors, while also providing a natural technique to handle error bars on physical measurements. While other hybrid system models can be extended to handle error bars in measurements, CLP(F) handles them naturally with no extra effort either in specifying them or in the calculation. A further advantage is that CLP(F) handles non-linear ODEs directly and soundly. CLP(F) also allows one to incrementally refine a model as one learns more about the physics of a system. At areas near a transition, one need not understand the details of the transition, but can simply bound the behaviour in that area, and get sound results. As one learns more about the physics, one can refine the model.

In addition CLP(F) is a very expressive language. It is easy to state problems involving finding values of the control parameters which result in specified behavior. Similarly, finding the time of state transitions is a simple matter of describing the transition as a constraint. Once the problem is stated, the underlying Prolog interpreter automatically solves it. Other ODE approaches often require an explicit binary search to find such times.

Limitations and Future Work

Currently there are several limitations on using CLP(F) to model hybrid systems. Each of them is an obvious possibility for future work: It would be helpful to develop more efficient interpreters for CLP(F) to handle very large complex systems. So far we have put almost no work into the efficiency of CLP(F), so there is a great deal of room for improvement here. We would like to extend this work to hybrid systems where the sensors are governed by PDEs rather than ODEs. Further efficiency improvements would come from developing primitive implementations of the `ode` procedures so that one does not need to use the full power of ACLP (and its accompanying inefficiencies).

We would like to develop more realistic models of hybrid systems using this approach.

CLP is not complete. If the search is bounded (e.g. there is a limit on the time parameter), then the entire search space will be traversed via backtracking and the incompleteness of Prolog is not an issue. The incompleteness of the CLP solver is however an issue whose consequence is that we do not know for certain whether any particular answer constraint actually contains a solution.

There is room for improvement in the heuristics of the underlying constraint solver. Since the CLP system guarantees soundness, extra attempts to narrow cannot introduce errors, but they do take time. Better heuristics could improve performance, both by reducing the running time and by narrowing the resulting intervals.

This paper has demonstrated that Analytic Constraint Logic Programming provides a promising approach to modeling hybrid systems by providing a program whose semantics precisely match the behavior of the hybrid system. Further research is needed to see if such an approach can be scaled up to real-life systems.

References

Acton, F. S. 1996. *Real computing made real: Preventing Errors in Scientific and Engineering calculations*. Princeton, New Jersey: Princeton University Press.

Adams, E., and Kulisch, U., eds. 1993. *Scientific Computing with Automatic Result Verification*. Academic Press.

Alur, R.; Courcoubetis, C.; Halbwachs, N.; Henzinger, T. A.; Ho, P.-H.; Nicollin, X.; Olivero, A.; Sifakis, J.; and Yovine, S. 1995. The algorithmic analysis of hybrid systems. *Theoretical Computer Science* 138:3–34.

Benhamou, F., and Older, W. J. 1997. Applying interval arithmetic to real, integer, and boolean constraints. *Journal of Logic Programming* 32(1):1–24.

Bohlender, G. 1996. Literature on enclosure methods and related topics. Technical report, Institut für Angewandte Matematik, Universität Karlsruhe, Postfach 6980, D-76128 Karlsruhe, Germany. http://www.uni-karlsruhe.de/∼Gerd.Bohlender/litlist.html an earlier version appeared in (Adams & Kulisch 1993).

Ciarlini, A. E., and Frühwirth, T. 2000. Automatic derivation of meaningful experiments for hybrid systems. In *ACM SIGSIM Conference on AI, Simulation and Planning (AIS '2000)*.

Deshpande, A.; Göllü, A.; and Semenzato, L. *The SHIFT Programming Language and Run-time System for Dynamic Networks of Hybrid Automata*. Department of Electrical Engineering and Computer Sciences; University of California at Berkeley, Berkeley, CA 94720. http://www.path.berkeley.edu/shift/doc/ieeshift.ps.gz.

Deville, Y.; Janssen, M.; and van Hentenryck, P. 2002. Consistency techniques in ordinary differential equations. *Constraints* 7(3):289–315.

Gupta, V.; Jagadeesan, R.; and Saraswat, V. 1996. `Hybrid cc`, hybrid automata and program verification. In Alur, R.; Henzinger, T. A.; and Sontag, E. D., eds., *Hybrid Systems III: Verification and Control*, volume 1066 of *LNCS*, 52–63. Springer Verlag.

Henzinger, T. A.; Horowitz, B.; Majumdar, R.; and Wong-Toi, H. 2000. Beyond HyTech: Hybrid systems analyis using interval numerical methods. In Lynch, N., and Krogh, B. H., eds., *Hybrid Systems: Computation and Control (HSCC 2000)*, volume 1790 of *LNCS*, 130–144. Springer Verlag.

Henzinger, T. A.; Ho, P.-H.; and Wong-Toi, H. 1998. Algorithmic analysis of nonlinear hybrid systems. *IEEE Transactions on Automatic Control* 43:540–554.

Hickey, T. J., and Wittenberg, D. K. 2003. Rigorous modeling of hybrid systems using interval arithmetic constraints. Technical Report CS-03-241, Brandeis University. http://www.cs.brandeis.edu/∼dkw/papers/cs03-241.pdf, accepted to HSCC 04.

Hickey, T. J. 2000. Analytic constraint solving and interval arithmetic. In *POPL'00 ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, 338–351. published as vol. 27 of SIGPLAN notices.

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.

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

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

Kuipers, B. J. 1993. Qualitative simulation: Then and now. *Artificial Intelligence* 59:133–140.

Lynch, N.; Segala, R.; Vaandrager, F. W.; and Weinberg, H. 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.

Lynch, N.; Segala, R.; and Vaandrager, F. 2001. Hybrid I/O automata revisited. In Benedetto, M. D. D., and Sangiovanni-Vincentelli, A., eds., *Hybrid Systems: Communication and Control*, volume 2034 of *LNCS*, 403–417. Springer Verlag.

Maler, O.; Manna, Z.; and Pnueli, A. 1991. From timed to hybrid systems. In de Bakker, J.; Huizing, C.; de Roever, W.; and Rozenberg, G., eds., *Real-Time: Theory in Practice*, volume 600 of *LNCS*, 447–484. Mook, The Netherlands: Rex Workshop.

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

Mosterman, P. J. 1999. An overview of hybrid simulation phenomena and their support by simulation packages. In Vaandrager, F. W., and van Schuppen, J. H., eds., *Hybrid Systems: Computation and Control*, volume 1569 of *LNCS*, 165–177. Springer Verlag.

Podelski, A. 2000. Model checking as constraint solving. In Palsberg, J., ed., *Proceedings of SAS'2000: Static Analysis Symposium*.

Smith, D. A., and Hickey, T. J. 1990. Partial evaluation of a CLP language. In Debray, S., and Hermenegildo, M., eds., *Proceedings of the 1990 North American Conference in Logic Programming*, 119–138.

Urbina, L. 1996. Analysis of hybrid systems in CLP($\mathcal{R}$). In Freuder, E. C., ed., *Principles and Practice of Constraint Programming – CP96*, volume 1118 of *LNCS*, 451–467. Springer Verlag.