Accurate Rigorous Simulation Should be Possible for Good Designs

Adam Duracz[1], Ferenc A. Bartha[2], Walid Taha[1]

[1]School of Information Technology, Halmstad University, Halmstad, Sweden {**Adam.Duracz, Walid.Taha**}**@hh.se**
[2]Department of Computer Science, Rice University, Houston TX, USA {**Ferenc.Bartha, Taha**}**@rice.edu**

The development of Cyber-Physical Systems benefits from better methods and tools to support the simulation and verification of hybrid (continuous/discrete) models. Acumen is an open source testbed for exploring the design space of what rigorous-but-practical next-generation tools can deliver to developers. Central to Acumen is the notion of rigorous simulation. Like verification tools, rigorous simulation is intended to provide guarantees about the behavior of the system. Like traditional simulation tools, it is intended to be intuitive, practical, and scalable. Whether these two goals can be achieved simultaneously is an important, long-term challenge.

This paper proposes a design principle that can play an important role in meeting this challenge. The principle addresses the criticism that accumulating numerical errors is a serious impediment to practical rigorous simulation. It is inspired by a twofold insight: one relating to the nature of systems engineered in the real world, and the other relating to how numerical errors in the simulation of a model can be recast as errors in the state or parameters of the model in the simulation. We present a suite of small, concrete benchmarks that can be used to assess the extent to which a rigorous simulator upholds the proposed principle. We also report on which benchmarks Acumen's current rigorous simulator already succeeds and which ones remain challenging.

I. Introduction

It is widely held that there is a need for new methods and tools to support model-based development using hybrid continuous/discrete systems models. Acumen [17] is an open source testbed for research into such methods and tools. Central to Acumen is a notion of *rigorous simulation* that is intended to combine the rigor of verification tools like Charon [3], KeYmaera [16], SpaceEx [8], and Flow* [5] with the ease of use of simulation tools like MATLAB [14] and Modelica [9]. Rigorous simulation computes an enclosure guaranteed to include the exact solution. As such, it is a kind of proof that the exact solution/simulation is contained in a particular set. Compared to more traditional mechanized theorem provers, rigorous simulation works in a simplistic, brute-force manner in that it largely follows the traditional methods of simulation. The key difference is that it tracks all possible forms of error.[1] Its similarity to traditional numerical algorithms offers hope that the run-time performance of rigorous simulation can be predictable, which would advance the ease-of-use goal. However, rigorous numerical computation, including rigorous simulation, is sometimes criticised as being infeasible in practice on account of the supposedly inevitable accumulation of large numerical error.

This work was supported by US NSF award CPS-1136099, the Swedish Knowledge Foundation (KK), The CERES Center, and VINNOVA (Dnr. 2011-01819).

[1]See for example Darulova and Kuncak for a discussion of different types of error [6].

A. Contributions

This paper proposes a design principle that addresses the above-mentioned criticism. The key insight underlying the principle is twofold:

1) Most man-made systems are designed to be stable and robust to variations in inputs and physical parameters.
2) In many situations, it should be possible to transform uncertainty about exact solutions into uncertainty about the state or parameters of the system modeled.

These observations together suggest that the following principle is both desirable and plausible:

> *Accurate rigorous simulation should be possible for good designs. In particular, enclosures computed by a rigorous simulation tool for the trajectories of a robust and stable hybrid system should be converging.*

This principle has been an important motivation for our own work, and we hope that it could be of value to others pursuing rigorous simulation or similar methods. We also hope that this work may spur efforts to formalize this principle as one or more properties about concrete rigorous simulation algorithms and implementations.

The principle allows us to concede that there are indeed cases in which the accumulation of errors can lead to diverging enclosures, thus providing little information. At the same time, it allows us to recognize that many important real-world systems for which engineers need better tools have characteristics that can alleviate this concern and render it irrelevant in many - if not most - practical situations.

B. Organization of the Rest of this Paper

After a brief review of the syntax and informal semantics for a core subset of Acumen (Section II), we present a suite of small, concrete benchmarks that can be used to assess the extent to which a rigorous simulator upholds this principle. The benchmarks are classified into discrete systems and timed systems (Section III); continuous systems (Section IV); and hybrid systems (Section V). As the benchmarks are presented, we identify those where Acumens current rigorous simulator [1] already succeeds, and those which still remain as challenges. We conclude with a summary of observations and directions for future work (Section VI).

II. A Core Subset of Acumen

In this section we briefly describe the emerging design for the syntax and semantics of the core subset of Acumen that will be discussed in the rest of this paper.

A. Syntax and Informal Semantics

The emerging core syntax for Acumen includes guarded equations, where guards are conditions and equations can specify either behaviors continuous with respect to time or discrete transitions discontinuous with respect to time. Expressions in equations can include standard functions on real numbers and derivatives of functions.

Values in core Acumen are functions of super-dense time (a real number and a natural number), and their co-domain is a real-number. We exclude from this core language some interesting features inessential to the main point of this paper. These include undirected equations, partial derivatives, dynamically created and terminated objects, strings, vectors, matrices, and visualisation constructs.

The Acumen implementation supports a *traditional simulation semantics* that uses non-validated numerical methods. This is the most complete and most widely used semantics for basic educational uses of Acumen. It has played a crucial role in allowing us to explore the language design space and to converge on an expressive, minimal syntax for modeling hybrid systems and on a formal (exact) semantics for solutions. Among other design choices, it allowed us to make an early decision to support the notion of super-dense time, first introduced in the verification literature [12], and later advocated by Liu, Matsikoudis and Lee [11]. This notion facilitates modeling of discrete subsystems with multiple idealized (zero time) and externally-observable internal transitions.

More relevant to the present work is that the implementation supports an *enclosure simulation semantics* intended to produce rigorous over-approximations (guaranteed upper and lower bounds) for all simulations [10]. This is the semantics that we would like Acumen, ultimately, to provide as the primary semantics. While this semantics is still a work in progress and is not defined for all constructs handled by the traditional one, our goal is to fully align the two.

The enclosure semantics supports the use of intervals (closed, compact, and connected sets on the reals) in models. As representations of sets (or, in computer science terminology, non-determinism), enclosures and intervals provide a tool for rigorous analysis of systems with uncertain parameters. This has been particularly important for collaborations with partners from the automotive industry [7], [13].

The enclosure semantics steps forward in time in a manner similar to traditional numerical simulators. The implementation supports variable step size. Currently, this is intended to reduce the uncertainty resulting from the treatment of events. The release associated with this paper uses fixed time step to make it easier to see how numerical simulation adds uncertainty. The implementation uses Lohner sets to represent enclosures. This is a standard representation that reduces the effects of wrapping while solving Ordinary Differential Equations (ODEs). Acumen has its own validated ODE solver [18], [19], [4], [15], which is built for portability and eventual formal verification. It is expected that it will be the subject of significant further development in the coming years. An integrator using Taylor series is used, which, in combination with Lohner sets, provides reasonable basic machinery for dealing with continuous segments of a simulation.

If a discrete assignment (basically, a reset map) becomes active, then it is performed and the entire model is checked again for discrete assignments. This process is repeated until a fixed point is reached. All equations in the model can be seen as guarded by some condition. Because enclosures and intervals are used for all values, it is possible that some guard conditions are not decidable. For example, this is the case when there is Zeno behavior. Acumen's method for dealing with Zeno behavior is described elsewhere [10]. However, it is possible that a guard condition is not decidable even in the absence of Zeno behavior. A more common situation is where the enclosure for an event straddles the start or the end of the current simulation step. In such a case, the state is split into two parts that safely over-approximate the true and false cases for the guard condition, and both parts are simulated. Such parts are currently kept separate until their evolutions have triggered the same sequence of guard conditions, at which point they are merged back into one.

So far, we have avoided introducing a static type system in the implementation. This choice is made to facilitate focus on implementation techniques and operational semantics, as well as to maintain a low entry barrier.

To familiarize the reader with Acumen's syntax and semantics, the following subsections present examples that illustrate how discrete, continuous, and hybrid systems are expressed.

B. Modeling Purely Discrete and Timed Systems

A purely discrete system that counts from 0 to 5 instantaneously[2] is expressed as follows:

```
model Disc0 () =
initially n = 0
always if n < 5 then n+ = n+1 noelse
```

The first line indicates that the model is called `Disc0` and that instances of this model take zero parameters. The equation following the keyword `initially` sets the initial value for the local variable `n` to 0. This is the *first* value for `n` at time 0. Because Acumen supports super-dense time, a variable can have a sequence of values at the same real-valued time instant. At all times greater than or equal to 0, one branch of the conditional or the other will be true. Most of the work done by this particular conditional, however, will be done at time 0: the first branch will be true until the condition `n<5` no longer holds. As long as that condition is true, the variable `n` is repeatedly assigned a new "next" value (written `n+` on the left-hand side of the equation) equal to one plus the current value of `n`. Thus `n` will have six different values (0, 1, .. 5) at time 0. Henceforth, it will remain at value 5.

This is a highly simplistic example illustrating how conditionals and reset maps (assignments for the next value of

[2]Meaning, with no progress of real-valued time

a variable, expressed as `n+ = ...`) allow us to express purely discrete computations. Nevertheless, it illustrates how the ability to express purely discrete computations ensures that the formalism can express "idealized" discrete computations that are treated as occurring in zero time. For example, this is how discrete computations are modeled in synchronous languages.

When the computations in such a model are on discrete values, the resulting enclosures should be *thin*, that is, contain a single value. This is the case for the enclosures that Acumen produces on this model.

We will use the term *timed system* to refer to a system that is mostly discrete but is triggered by an analog clock, such as a timed automata [2]. Such systems are, in fact, hybrid systems, and modeling them is described in the relevant subsection below.

C. Modeling Purely Continuous Systems

A purely continuous system with a single variable that is linearly increasing with time can be expressed as follows:

```
model Cont0 () =
initially t = 0, t' = 1
always t' = 1
```

In contrast to the first model above, this system refers to the derivative of a variable (denoted by `'`) rather than the next value of a variable. The equation `t'=1` is a differential equation that says that the time derivative of `t` is always equal to 1. Thus, the resulting simulation for `t` starts at value 0 at time 0 and has the same value as time lapsed since instantiation of this particular model. Expressing continuous systems is important for modeling physical components and phenomena. As some other examples in this paper will illustrate, more interesting dynamics can be expressed by replacing the right-hand side of the equation with other expressions.

Acumen's rigorous simulator uses only validated (rigorous) numerical integration to solve all ODEs such as the one above. In particular, no effort is made to do symbolic integration or to find closed form solutions to certain classes of differential equations.[3] As a result, a certain amount of error can accumulate in each simulation time step. Consequently, as simulation time increases, the size of the enclosure for the value of `t` can gradually increase. Despite this phenomenon, examples presented later in this paper will show how systems can have contracting enclosures even with large uncertainty about local timers such as `t` presented here.

D. Modeling Hybrid Systems

A hybrid continuous/discrete system mixes both continuous dynamics and discrete computations and transitions. The following example can be seen as a minimal prototype for such systems:

```
model Hybrid0 () =
initially t = 0, t' = 1
always if t<1 then t' = 1 else t+ = 0
```

Conditionals or guards as represented by the `if` statement are commonly viewed as a mechanism for *event detection*. In this case, the event is represented by the condition being true or false. In general, an event can be seen as happening when the condition changes from true to false or vice versa. In this example, the system allows `t` to grow at a constant rate until it reaches the value of 1. As soon as this value is reached, the second branch becomes active, and the value of `t` is suddenly reset to 0. The result of the simulation should therefore be a saw-tooth shaped pattern.

This is another system that is meta-stable rather than stable in the sense of, say, exponential stability. Thus the enclosure can be expected to accumulate error over time and grow in size. This happens in Acumen. This is significant because the resulting sawtooth pattern for `t` provides a natural basis for a simple discrete clock. However, as we will see with examples later on in this paper, even complete uncertainty about the value of the clock does not prevent us from knowing with perfect precision the behavior of the system after a predetermined finite time.

E. Remark on Locally Discrete Systems

There is a difference between how discrete behaviors should be made for variables for which a derivative has been declared (in the `initially` section) and ones for which it has not. The first example above shows us how to express such behavior when no derivative is introduced. If one is introduced, it becomes necessary to specify the value for the derivative when no discrete change is taking place. For example, if there is no change between discrete changes, then the derivative should be set to 0. This is illustrated by the following example:

```
model DiscHybrid0 () =
initially t = 0, t' = 0
always if t<5 then t+ = t+1 else t' = 0
```

Without such a statement, a variable for which a derivative was declared is considered under-specified.

III. Discrete and Timed Systems

This section presents examples of discrete systems where enclosures should converge, despite the presence of considerable uncertainty not only from numerical calculations but also from potential variability in system parameters.

For the purposes of this paper, we will view an enclosure as *converging* if it reaches a fixed point where it is a subset of the enclosure at a previous point in simulation time. For simplicity of implementation we will further approximate this condition by testing set containment only in the enclosure of the simulation step that immediately precedes it.

A. Iteration and Nested Loops

Iteration can be expressed naturally in core Acumen. For example, the computation for factorial of 5 can be expressed as follows:

[3]Such techniques can be sound optimizations, but it is important that the design of the tools does not rely on them, since for general applications such special cases are not always sufficient.

```
model Disc1 () =
initially
 n = 5, a = 1
always
 if n > 0 then n+ = n-1, a+ = a*n noelse
```

A more interesting nested loop can be expressed as follows:

```
model Disc2 () =
initially i = 1, i_max = 3,
          j = 1, j_max = 4,
          k = 1, k_max = 5,
          a = 0
always
 if i<=i_max then
   if j<=j_max then
     if k<=k_max then
       a+ = a+1, k+ = k+1
     else (j+ = j+1, k+ = 1)
   else (i+ = i+1, j+ = 1)
 noelse
```

This example simply counts the number of iterations into the variable `a`. Because the computations are discrete (no integration, only representable numbers), Acumen computes a thin (that is, single value) enclosure for the result.

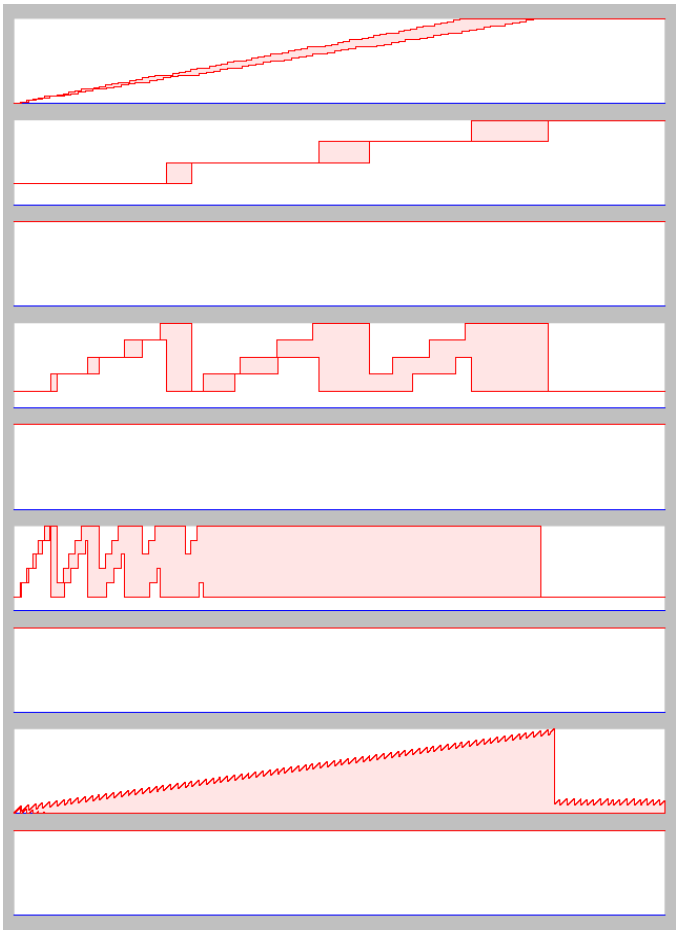
Without really introducing any interesting dynamics, we often want to model computations as taking some time to compute. This can be achieved using a simple clock, such as the one introduced previously, to produce the following model:

```
model Disc3 () =
initially i = 1, i_max = 3,
          j = 1, j_max = 4,
          k = 1, k_max = 5,
          a = 0, t = 0, t' = 1
always
 if t>0.1 then
   t+ = 0,
   if i<=i_max then
     if j<=j_max then
       if k<=k_max then
         a+ = a+1, k+ = k+1
       else (j+ = j+1, k+ = 1)
     else (i+ = i+1, j+ = 1)
   noelse
 else t'= 1
```

Acumen's Standard Plot and How to Read it: By default, Acumen produces a standardized plot for a simulation. The interactive graphical user interface allows the user to zoom in and point to parts of the enclosure to read the time and value intervals. Labels are omitted to reduce clutter. To read the plot, the reader should be aware of the following conventions:

- Within the gray boundary, each white rectangle represents the plot for a particular variable.
- From top to bottom, variables appear in alphabetical order, with derivative variables appearing directly after the variable.
- The vertical scale is normalized to fit the minimum and maximum values for each plot during the simulation time.
- By default, the simulation is for 10 seconds. The horizontal axis represents time, with time 0 at the left end and time 10 at the right end. For converging enclosures the entire time is shown. For diverging enclosures, the simulation is stopped at an earlier time to provide a more informative visualization. In such cases, the time at which the simulation is stopped is indicated in the text.
- All plots are generated using a fixed time step of 2^{-6}.

The plot for the model above is as follows:

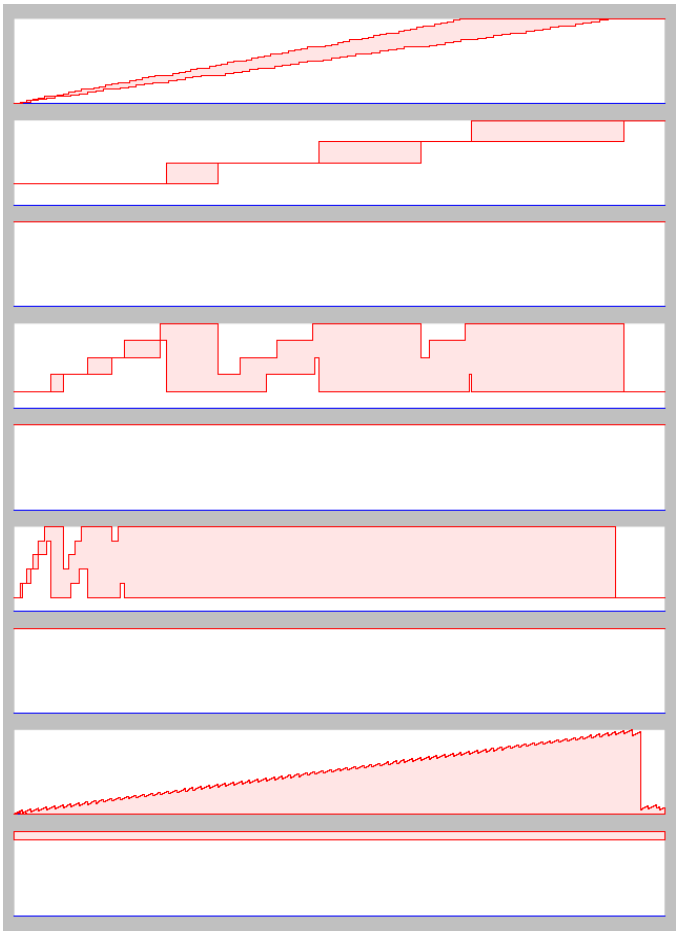


Here, instead of getting the answer 60 in the variable `a` at time 0, we get to see the evolution of the value of `a` over time, as well as the evolution of the values of all three counter variables (`i`, `j`, and `k`). Despite the accumulation of numerical errors in the clock variable `t`, after a finite amount of time the value of `a` becomes exactly 60. In the interactive development environment, this is confirmed by hovering the mouse over the graph so that the exact value of the interval is displayed textually. The unusual initial increase in the enclosure for the variable `t` (the increasing sawtooth) is due to an over-approximation that results from Acumen's current method for merging branched states. This is an implementation artifact uncovered during this work. It has no bearing on the main point of this paper, but it is something that can cause convergence to fail unnecessarily on some examples. We plan to address this issue in future work.

Returning to the models, we can consider whether convergence continues to hold if some parameters of the system are not known exactly. For example, the following model specifies uncertainty in the rate of the clock and the timing of the clock reset:

```
model Disc3i2 () =
... // same as Disc3
always
... // same as Disc3
 else t'= [0.90 .. 1.0]
```

The expression `[0.90 .. 1.0]` denotes an interval literal that describes that `t'` can be as little as `0.90` or as large as `1.0`. The plot for the model above is as follows:



For this model and the previous three, Acumen produces enclosures that first expand until a certain time value is reached. At that time value, all enclosures except that of `t` contract to a single value. Such convergence of enclosures on discrete systems with uncertainty about timing can provide a useful tool for estimating worst-case execution times, given a particular computational platform for which parameters are known only with a modest precision. Technically, achieving single-step containment requires disabling branch merging for all variants.

B. Finite Impulse Response (FIR) Filter

As a minimal example of digital signal processing (DSP) code, we consider a basic finite impulse response (FIR) filter:

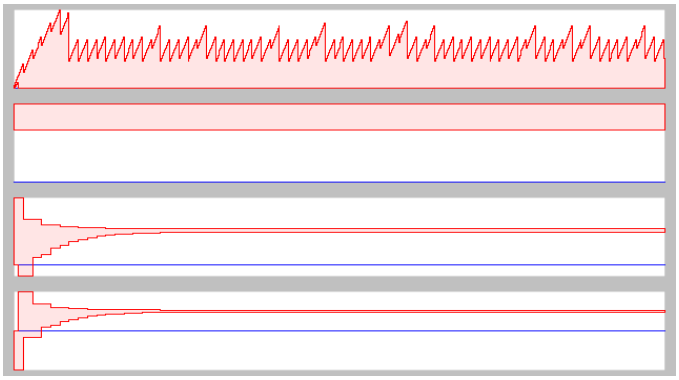
```
model Disc4 () =
initially x_0 = 10, x_1 = 0,
          t = 0, t' = 1
always
 if t>0.1 then
   t+ = 0,
   x_1+ = x_0,
   x_0+ = 0.33*x_0 + 0.33*x_1 + 0.33*5
 else t'= 1
```

This is a second order filter that responds to a signal with values 10, 0, and then 5 thereafter. The enclosure produced by Acumen seems to quickly converge towards 5 for the value of `x_0`, even though the clock variable `t` continues to accumulate uncertainty.

To further evaluate the ability of rigorous simulation to demonstrate the robustness of this filter, we can model uncertainty about the value of multiple parameters, including initial values, clock speed, transition timing, and the precision of the calculation of the multiplications (reflected by imprecision in the representation of coefficients) as follows:

```
model Disc4i5 () =
initially x_0 = [0.0 .. 10.0],
          x_1 = [-10.0 .. 0],
          t = 0, t' = 1
always
 if t>[0.09 .. 0.11] then
   t+ = 0,
   x_1+ = x_0,
   x_0+ = [0.33 .. 0.34] * x_0 +
          [0.33 .. 0.34] * x_1 +
          [0.33 .. 0.34] * 5
 else t'= [0.8 .. 1.2]
```

The plot for this model is as follows:



Acumen's results for this model converge towards 5, although there is a notable "error margin" around 5 that the enclosures stay outside. This is encouraging in that it shows that the computed behavior is robust. However, this plot should inspire a modeler/designer to dig deeper to better understand the most significant causes for this "error margin".

Both examples presented in this section illustrate how enclosures can grow during a simulation of a discrete system and still converge at later parts, thus providing computational proof that the particular system being considered is stable and robust to a given degree of variability, in the exact values of key parameters within the given level of uncertainty. Interestingly, in these examples, single-step containment was not achieved except in the final (most general) example. It was necessary to disable branch merging, and the earlier examples generated too many branches.

IV. Continuous Systems

Uncertainty about the initial state of the system described by the second model above can be expressed as follows:

```
model Cont0i0 () =
initially t = [0.0 .. 1.0], t' = 1
always t' = 1
```

The enclosure for this model is not contracting. Instead, we get a wide band representing all possible solutions starting between 0 and 1.0 and increasing at a constant rate of 1. The non-convergent nature of the enclosure is to be expected, as the underlying system is not stable in the sense of, for example, converging exponentially to a given value.

A. A First Order Linear System

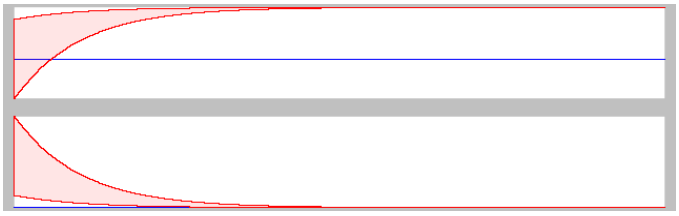
A prototypical example of an exponentially stable system can be expressed as follows:

```
model Cont1 () =
initially x = 0, x' = 1
always x' = 1-x
```

Uncertainty about the initial value for such a system can be expressed as follows:

```
model Cont1i0 () =
initially x = [-0.75 .. 0.75], x' = 1
always x' = 1 - x
```

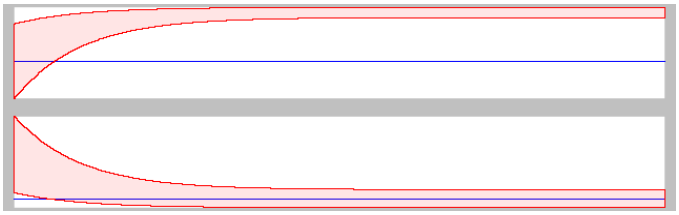
and has the following plot:



The plot shows that the enclosure for `x` converges towards 1. Even if we add uncertainty about the value of `x` used in the equation, as expressed in this model, the enclosure appears to continue to converge:

```
model Cont1i1 () =
initially x = [-0.75 .. 0.75], x' = 1
always x' = 1 - [0.9 .. 1.1] * x
```

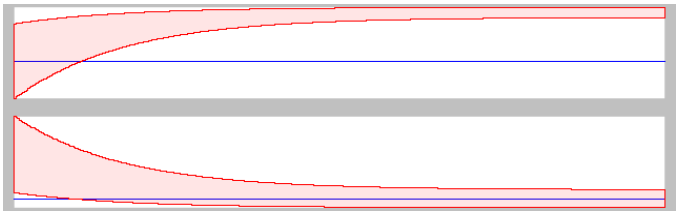
The plot for this enclosure is as follows:



Convergence is towards an "error margin" around the value of 1 because the precise value depends on the coefficient of `x`. If there is explicit uncertainty about the target, as expressed in this model, the resulting enclosure has similar behavior:

```
model Cont1i2 () =
initially x = [-0.75 .. 0.75], x' = 1
always x' = [0.9 .. 1.1] - x
```

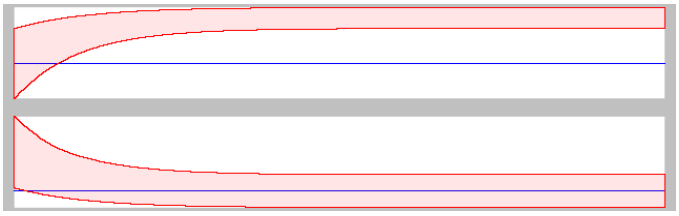
The plot for this enclosure is as follows:



The following model combines all three types of uncertainty:

```
model Cont1i3 () =
initially x = [-0.75 .. 0.75], x' = 1
always x' = [0.9 .. 1.1] -
            [0.9 .. 1.1] * x
```

The following plot shows that we still have convergence:



It also seems that the error margin becomes about as big as the sum of the two previous error margins. As in the FIR example, it will be up to the modeler/designer to determine the implications of this error margin on the success of the design.

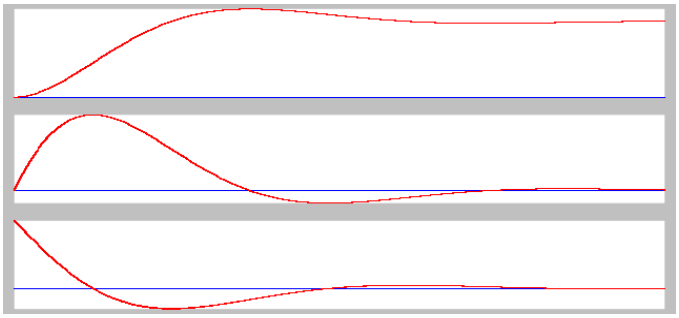
Single-step containment was achieved for all variants with the default settings.

B. A Second Order Linear System

A second order system exposes more subtleties and some challenges to convergence. Consider the following basic model:

```
model Cont2 () =
initially x = 0, x' = 0, x'' = 1
always x'' = (1-x)-x'
```

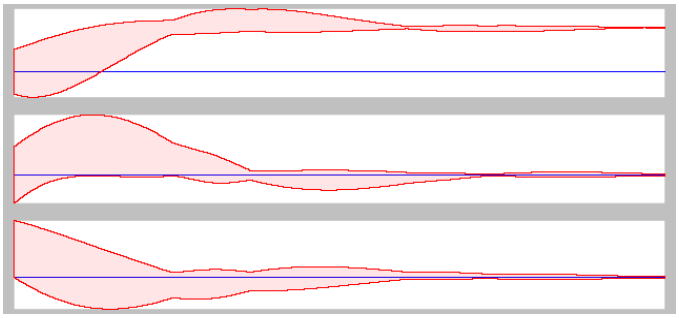
The plot for this for model is as follows:



This enclosure appears to converge. We can add uncertainty about the initial values as follows:

```
model Cont2i1 () =
initially x = [-0.5 .. 0.5],
          x' = [-0.5 .. 0.5], x'' = 1
always x'' = (1 - x)-x'
```

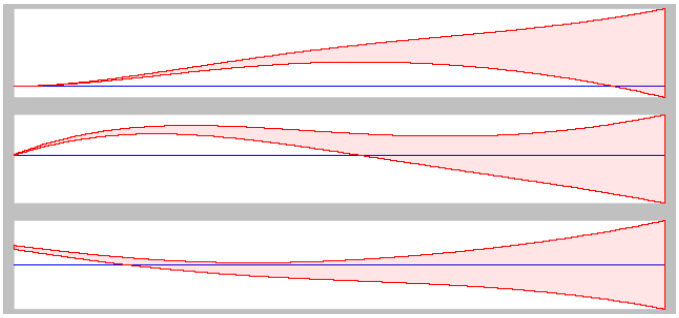
The plot for this model is as follows:



The enclosure is more interesting and still appears to converge. Next, we may consider introducing uncertainty about the target value as follows:

```
model Cont2i2_ () =
initially x = 0, x' = 0, x'' = 1
always x'' = ([0.90 .. 1.1] - x) - x'
```

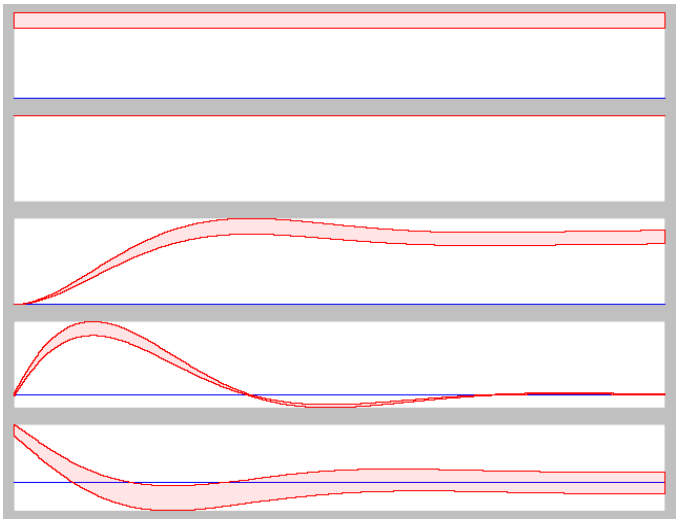
The plot for this model up to time 5 seconds is as follows:



Unfortunately, this model is not converging. But can a slight reformulation help convergence? Consider the following model:

```
model Cont2i2 () =
initially x = 0, x' = 0, x'' = 1,
          a = [0.9 .. 1.1], a' = 0
always x'' = (a - x) - x', a' = 0
```

It has the following plot:



With default settings the enclosure for the model still eventually diverges. However, adding a basic error redistribution method does achieve single-step containment for all variants above except the last one (`2i2`). Thus, seemingly minor changes can affect the impact of redistribution.

Now let us consider introducing uncertainty in the gain in the feedback in this equation:

```
model Cont2i3__ () =
initially x = 0, x' = 0, x'' = 1,
          b = [0.9 .. 1.1], b' = 0
always x'' = b * (1 - x) - x', b' = 0
```

With default settings, this has a divergent enclosure. Similarly, even a named coefficient, as expressed here, yields a divergent enclosure:

```
model Cont2i4__ () =
initially x = 0, x' = 0, x'' = 1,
          c = [0.5 .. 1.5], c' = 0
always x'' = (1 - x) - c * x', c' = 0
```

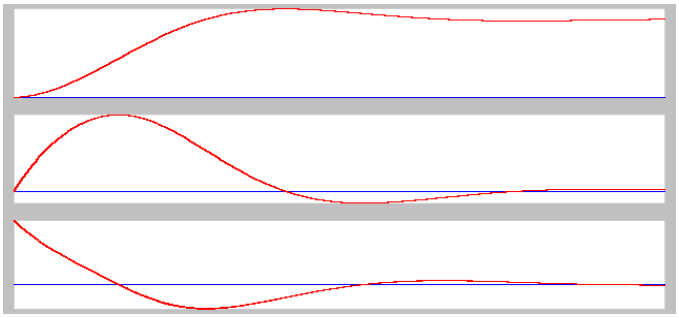
Enabling error redistribution removes the apparent divergence, but is not sufficient to achieve single-step containment.

C. A Second Order Non-Linear System

A classic non-linear equation is that of a pendulum:

```
model Cont3 () =
initially x = 0, x' = 0, x'' = 1
always x'' = sin(pi/2-x)-x'
```

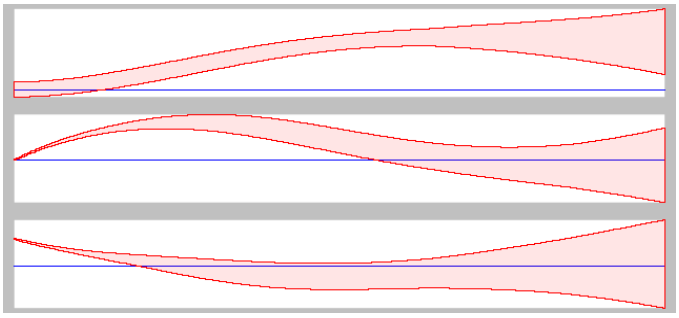
This kind of non-linearity is typical in 2- and 3-dimensional classical mechanics. The plot for this model is as follows:



The enclosure for this system appears to converge. However, consider a variant with even minor uncertainty about the initial conditions:

```
model Cont3i1_ () =
initially x =[-0.05 .. 0.05],
          x' = 0, x''=1
always x'' = sin(pi/2-x)-x'
```

The resulting enclosure diverges:



We can recover single-step containment by error redistribution in all cases where we have uncertainty in one parameter at a time. But when we have uncertainty in all parameters, this fails and we observe divergence.

V. Hybrid Systems

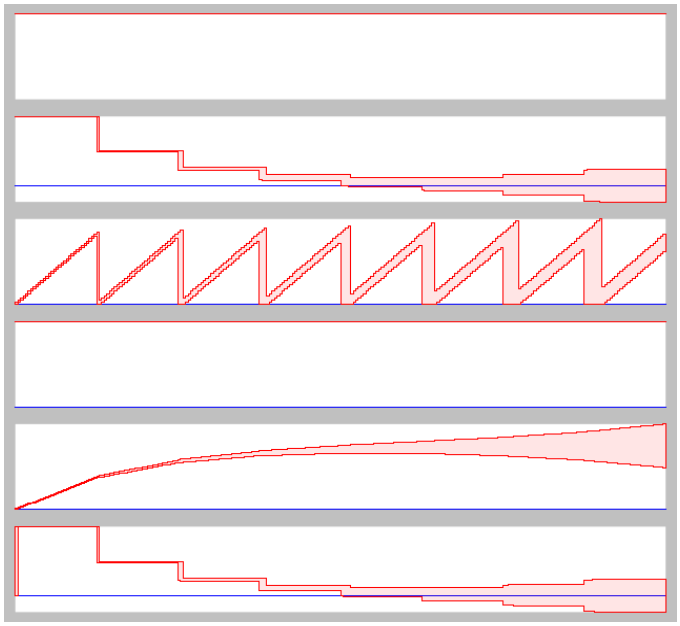
In this section we present observations and benchmarks relating to hybrid systems. The benchmarks illustrate both how quickly combining the continuous and discrete can compound complexities for achieving convergence, and how it can also offer some new opportunities for facilitating it.

A. Discretized Sensing/Actuation

The following model represents a situation where the output of a controller is not written continually to the system being controlled but rather in a sampled manner:

```
model Cont4hd_ () =
initially x = 0, x' = 0, i = 0,
          o = 1, t = 0, t' = 1
always
  x' = o,
  if t>0.5
    then o+ = 1-x, t+ = 0
  else t'=1
```

The plot for this model is as follows:



In the current implementation the enclosure for this system diverges. Our current hypothesis is that the cause is over-approximations in event handling. However, further investigation is needed to confirm this.

B. Zeno Systems

A final and important type of system that is useful for evaluating the extent to which an implementation realizes the proposed principle is Zeno systems [10]. Zeno behavior is known to be a phenomena unique to hybrid systems that naturally arises in models of mechanical systems, control systems, and others.

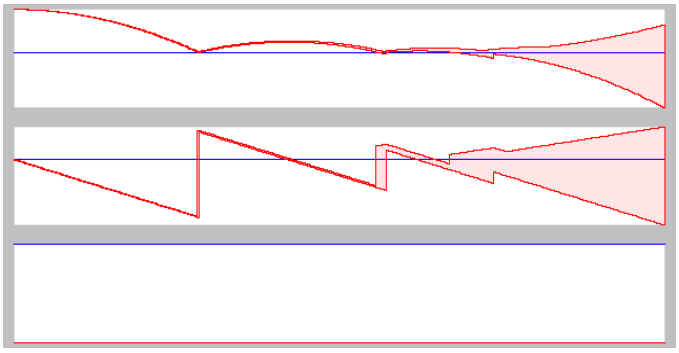
A classic example of a model that exhibits Zeno behavior is a bouncing ball:

```
model Hybrid1 () =
initially
  x = 10, x' = 0, x'' = -10
always
  claim x>=0,
  if x <= 0 && x' < 0
  then x'+ = -x'/2
  else x'' = -10
```

The variable `x` represents the height of a ball that falls to the ground. Upon impact with the ground, the ball bounces (switches direction) with half the speed. The `claim` statement is essentially syntactic sugar for a conditional that leads to blocking the system (having no solution) in the second branch. Analysis of the model would reveal that the bouncing speed becomes zero after a finite amount of time known as the Zeno

time. However, the ball would have to bounce an unbounded number of times before reaching this point.

In previous work [10] we were able to show that it is possible to simulate such models past the Zeno point using a fixed-point computation on enclosures during simulation. However, such enclosures are not necessarily converging past the Zeno point. They generally have the following form:

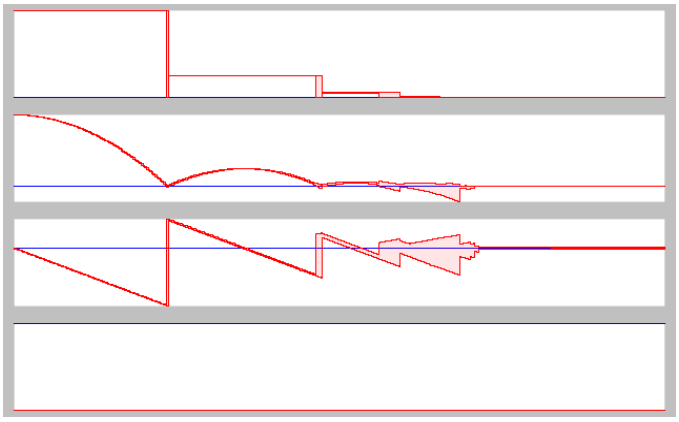


This enclosure is already interesting in that it is going beyond the Zeno point: traditional methods either loop infinitely or skip some events, sacrificing rigour.

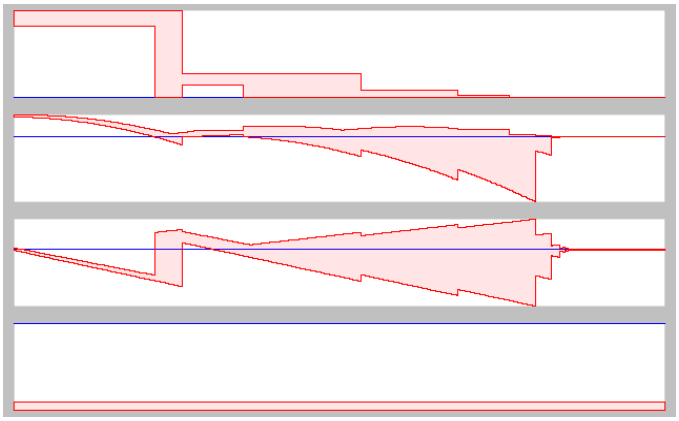
To get convergence in such situations, it seems necessary for the user to enrich the model with additional information that a powerful theorem prover could possibly infer, but is not self-evident. For the system described, the addition would be an explicit model of the energy of the bouncing ball, and an explicit statement of the relation between the speed of the ball and the energy at a given time. This model is expressed as follows:

```
model Hybrid2 () =
initially
  x = 10, x' = 0, x'' = -10,
  e = 0*0/2 + 10*10
always
  claim x >= 0,
  claim e == x'*x'/2 + 10*x,
  if x <= 0 && x' < 0
  then x'+ = -x'/2, e+ = [0 .. 0.25]*e
  else x'' = -10
```

It results in the following enclosure:



The enclosure converge, and adding a degree of uncertainly to virtually all parameters in this model does not seem to interfere with this convergence.



We note that this model cannot be simulated with branch merging disabled. Zeno systems generate a large number of branches that necessitate frequent merging.

VI. Conclusions

This paper articulates a principle for the design of rigorous simulation tools. The principle addresses the criticism that the accumulation of numerical errors can be a serious impediment to making such tools practical. It reflects a twofold insight: part relating to the nature of systems engineered in the real world, and another relating to how accumulating numerical errors can be dynamically recast as modeled errors. If the system modeled is robust and stable it should absorb the latter type of error, enabling more accurate simulation. We present a suite of small, concrete benchmarks that can be used to assess the extent to which a rigorous simulator upholds this principle. We also identify the benchmarks where Acumen's current rigorous simulator [1] already succeeds in, and which ones remain as goals.

Figure 1 summarizes our observations about Acumen's performance on these benchmarks. In some cases, Acumen already realizes the principle. In others, preliminary experiments suggest that an explicit error redistribution strategy, along with more precise branch merging, can lead to improvement. In others still, adding additional constraints in the models exhibiting Zeno behavior helps. But there are also cases, such as those of nonlinear ODEs and hybrid systems with delay (sample & hold), that still pose a challenge.

More broadly, there are practical challenges to testing convergence. First, here we simplified testing of convergence to comparing only with the last simulation step. A converging system can be periodic, and detecting such cases requires searching the entire history. Second, it was necessary to disable adaptive stepping to accelerate the accumulation of numerical errors. This leaves the possibility that enabling adaptive stepping can itself be another way to achieve convergence. Third, and maybe most importantly, convergence can depend on the size of the uncertainty. One way to address this problem may be to develop methods for finding a maximal set of uncertainties (under a suitable norm) for which the enclosure converges.

We hope that this work builds a good case for the value of such a principle in building practical tools, and encourages further exploration of this idea. More generally, we hope it stimulates more discussion on design principles that can help

Type	Feature	Model Uncertainty	Benchmark	Convergence (A/B)
Discrete	Loop	-	`Disc1, Disc2`	Y/Y
Timed	Loop	-	`Disc3`	–/Y
		Rate, or/and reset	`Disc3i0 ... 2`	–/Y
	FIR	Rate, reset, arithmetic, or IVs	`Disc4i0 ... 4`	–/–
		Rate, reset, arithmetic, and IVs	`Disc4i5`	–/Y
Continuous	Linear, 1st ord.	-	`Cont1`	Y/Y
		IV, gain, or/and target	`Cont1i0 ... 3`	Y/Y
	Linear, 2nd ord.	-	`Cont2`	N/Y
		Various	`Cont2i0,1,2_,3_,4_`	N/Y
		Various	`Cont2i2,3__,4__`	N/–
	Nonlinear, 2nd ord.	-	`Cont3`	N/Y
		IVs, target, or gains	`Cont3i1_ ... 5_`	N/Y
		IVs, target, and gains	`Cont3i6_`	N/N
Hybrid	Linear, 1st ord.	-	`Cont4`	–/–
	Sample & pass	-	`Cont4h_`	–/–
	Sample hold	-	`Cont4hd_`	N/N
	BB	-	`Hybrid1`	N/N
	BB w/ energy	-	`Hybrid2`	Y/Y
		IVs, all param's	`Hybrid2i1 ... 5`	Y/–

Fig. 1. Enclosure convergence of primary dynamics in benchmarks as demonstrated using Acumen's most recent rigorous simulator. Shorthands: A = Without error redistribution. B = With error redistribution and branch merging disabled. Y = Single-step containment confirmed. "–" = No apparent divergence (up to time 50 seconds model time). N = Divergence observed. FIR = Finite impulse response. IV = Initial value. BB = Bouncing Ball.

ensure the practical usability and accessibility of rigorous simulation tools.

VII. Acknowledgements

We would like to thank Kevin Atkinson, Paul Brauner, Jan Duracz, Fei Xu, and Yingfu Zeng for their contributions to the development of Acumen and the enclosure semantics. We would also like to thank Aaron Ames, Michal Konečný, Veronica Gaspes, and Warwick Tucker for valuable discussions about Zeno behavior, validated numerics, and numerical integration. Amin Farjudian, Mark Stephens, and Masoumeh Taromirad provided valuables comments on a draft of this paper.

VIII. References

[1] Acumen release 2016_03_22. Benchmarks can be found in the folder `examples/05_language_research/26_convergence` in the distribution. http://bit.ly/acumen-snr-2016.

[2] Rajeev Alur and David L Dill. A theory of timed automata. *Theoretical computer science*, 126(2):183–235, 1994.

[3] Rajeev Alur, Radu Grosu, Yerang Hur, Vijay Kumar, and Insup Lee. Modular specification of hybrid systems in charon. In *Hybrid Systems: Computation and Control*, pages 6–19. Springer, 2000.

[4] Ferenc A Bartha. *Computer-aided proofs and algorithms in analysis*. PhD thesis, The University of Bergen, 2013.

[5] Xin Chen, Erika Ábrahám, and Sriram Sankaranarayanan. Flow*: An analyzer for non-linear hybrid systems. In *Computer Aided Verification*, pages 258–263. Springer, 2013.

[6] Eva Darulova and Viktor Kuncak. Trustworthy numerical computation in scala. In *Proceedings of the 2011 ACM International Conference on Object Oriented Programming Systems Languages and Applications*, OOPSLA '11, pages 325–344, New York, NY, USA, 2011. ACM.

[7] Adam Duracz, Henrik Eriksson, Ferenc Ágoston Bartha, Yingfu Zeng, Fei Xu, and Walid Taha. Using rigorous simulation to support ISO 26262 hazard analysis and risk assessment. In *12th IEEE International Conference on Embedded Software and Systems*, pages 1093–1096. IEEE, 2015.

[8] Goran Frehse, Colas Le Guernic, Alexandre Donzé, Scott Cotton, Rajarshi Ray, Olivier Lebeltel, Rodolfo Ripado, Antoine Girard, Thao Dang, and Oded Maler. SpaceEx: Scalable verification of hybrid systems. In *Computer Aided Verification*, pages 379–395. Springer, 2011.

[9] Peter Fritzson. *Principles of object-oriented modeling and simulation with Modelica 2.1*. John Wiley & Sons, 2010.

[10] Michal Konečnỳ, Walid Taha, Ferenc A Bartha, Jan Duracz, Adam Duracz, and Aaron D Ames. Enclosing the behavior of a hybrid automaton up to and beyond a zeno point. *Nonlinear Analysis: Hybrid Systems*, 20:1–20, 2016.

[11] Xiaojun Liu, Eleftherios Matsikoudis, and Edward A Lee. Modeling timed concurrent systems. In *Concurrency Theory*, pages 1–15. Springer, 2006.

[12] Oded Maler, Zohar Manna, and Amir Pnueli. From timed to hybrid systems. In *Real-time: theory in practice*, pages 447–484. Springer, 1992.

[13] Jawad Masood, Roland Philippsen, Jan Duracz, Walid Taha, Henrik Eriksson, and Christian Grante. Domain analysis for standardised functional safety: a case study on design-time verification of automatic emergency braking. pages 845–854. Royal Netherlands Society of Engineers (KIVI), 2014.

[14] MATLAB. *version 7.10.0 (R2010a)*. The MathWorks Inc., Natick, Massachusetts, 2010.

[15] Nedialko S Nedialkov, Kenneth R Jackson, and George F Corliss. Validated solutions of initial value problems for ordinary differential equations. *Applied Mathematics and Computation*, 105(1):21–68, 1999.

[16] André Platzer and Jan-David Quesel. KeYmaera: A hybrid theorem prover for hybrid systems (system description). In *Automated Reasoning*, pages 171–178. Springer, 2008.

[17] Walid Taha, Adam Duracz, Yingfu Zeng, Kevin Atkinson, Ferenc A. Bartha, Paul Brauner, Jan Duracz, Fei Xu, Robert Cartwright, Michal Konečný, Eugenio Moggi, Jawad Masood, Pererik Andreasson, Jun Inoue, Anita Sant'Anna, Roland Philippsen, Alexandre Chapoutot, Marcia O'Malley, Aaron Ames, Veronica Gaspes, Lise Hvatum, Shyam Mehta, Henrik Eriksson, and Christian Grante. Acumen: An open-source testbed for cyber-physical systems research. In *Proceedings of EAI International Conference on CYber physiCaL systems, iOt and sensors Networks (CYCLONE), 2015*. EAI, 2015.

[18] Warwick Tucker. *Validated numerics: a short introduction to rigorous computations*. Princeton University Press, 2011.

[19] Piotr Zgliczynski. C 1 Lohner algorithm. *Foundations of Computational Mathematics*, 2(4):429–465, 2002.