Opportunity in Embracing Imperfection: Is simulation the real thing?

Pieter J. Mosterman
Senior Research Scientist
Design Automation Department

Adjunct Professor
School of Computer Science

McGill
In your opinion, what lasting legacy has YACC brought to language development?

YACC made it possible for many people who were not language experts to make little languages (also called domain-specific languages) to improve their productivity. Also, the design style of YACC - base the program on solid theory, implement the theory well, and leave lots of escape hatches for the things you want to do that don't fit the theory - was something many Unix utilities embodied. It was part of the atmosphere in those days, and this design style has persisted in most of my work since then.

http://news.idg.no/cw/art.cfm?id=094E3B6E-17A4-0F78-311509693E8E95C1
Opportunity in Embracing Imperfection: Is simulation the real thing?

Pieter J. Mosterman
Senior Research Scientist  
*Design Automation Department*

Adjunct Professor  
*School of Computer Science*

McGill
The importance of computation

Together with theory and experimentation, computational science now constitutes the “third pillar” of scientific inquiry,
Resolved, That the House of Representatives—

3) encourages the expansion of modeling and simulation as a tool and subject within higher education;

4) recognizes modeling and simulation as a National Critical Technology;
Agenda

- Outline
- Model-Based Design
- Problem statement
- A solution approach
- Outlook
The science in engineering a system

experimentation

Discard detail but keep pertinent behavior

theory
The science in engineering a system

Where is the heat?

But often no analytical solution
The science in engineering a system

- experimentation
- theory
- computation

A computational solution
The science in engineering a system

experimentation  theory  computation

A computational solution

Not much heat at the nozzles … let’s change the material …
The science in engineering a system

experimentation
computation

Not much heat at the nozzles … let's change the material …
Computational methods to add more detail

Same computational approximation
Information beyond what is in a first principles model
Computational methods to add more detail

Same computational approximation
Model quality

- Approximation
- Information
- Analytic
- Computation
- Decreasing error
Model quality

- Model quality
- Approximation
- Information
- Error level
- Decreasing error
- Computation
- Analytic
- Error level
Computational methods are not that mature
Computational methods are not that mature

“… engineers used Crater during STS-107 to analyze a piece of debris that was at maximum 640 times larger in volume than the pieces of debris used to calibrate and validate the Crater model.”

Technology maturation: a comparison

Conservative  Cautious improvement  ... till failure

1849  1883  1940

Széchenyi Chain Bridge  Tacoma Narrows Bridge  Brooklyn Bridge

Technology maturation: a comparison

Conservative

1849

Széchenyi Chain Bridge

Cautious improvement

1883

Tacoma Narrows Bridge

...till failure

1940

Brooklyn Bridge

Firm methodology

1998

Akashi-Kaikyō Bridge

“Computational Science Demands a New Paradigm,” Douglass E. Post and Lawrence G. Votta, Physics Today, 2005
Premise

- Approximation is **not** the culprit
  - Model-Based Design is successfully exploiting computation
  - But still very *ad hoc*; lots of testing required

- Embrace the imperfection!
  - Create better models **without** reducing the approximation
  - Use computational methods to:
    - Enhance model information
    - Enhance domain information
    - Analyze and design complex execution engines
  - Requires precise definition of the execution semantics
    - Differential equations, difference equations, discrete event, etc.
    - Approximations

- Treat computation as **equal** to experiment and theory
Agenda

- Outline
- Model-Based Design
- Problem statement
- A solution approach
- Outlook
Design of an engineered system
Increasingly more detail
System behavior
Simulation studies
Model-Based Design
Model-Based Design

Host stack

Target stack
Executable specifications
Model elaboration
Automatic code generation

Elaborate

Synthesize

Host stack

Target stack
Model-Based Design

- Raises level of abstraction
- Enables continuous testing

Explore
Verify
Test

Elaborate

Explore
Verify
Test

Synthesize

Explore
Verify
Test

Host stack

Target stack

Model-Based Design
Mapping an application

- Single application
- General purpose

Application to be implemented

Application specific

Technology divergence
Mapping an application

- Single application
- General purpose
- Application specific
Mapping an application

Compile technology into application

Single application

General purpose

Application specific
Mapping an application
Mapping an application

- Reuse shared transformation technology
- Single application
- General purpose
- Application specific
Mapping an application

- Modular timing engine to enable reuse

- Single application

- General purpose

- Application specific
Agenda

- Outline
- Model-Based Design
- Problem statement
- A solution approach
- Outlook
Computer Automated Multiparadigm Modeling for technology reuse

- A syntax, a semantic domain, and a mapping

For graphical syntax, often a meta model is used
Define semantics as a syntactic transformation—semantic anchoring

- Target semantic domain must be subsumed
Modeling a model transformation
Modeling a model transformation
Modeling a model transformation
Model-Based Design
Model-Based Design
Multi-domain models comprise many formalisms …

- Can we develop a unifying semantic domain?
Multi-domain models comprise many formalisms …

- Can we develop a unifying semantic domain?
Modeling a physical system

From first principles …

Hooke’s Law:

\[ F = -\frac{x - x_0}{C} \]

Newton’s Second:

\[ F = ma \]

A bit of calculus:

\[ a(t) = \frac{dv(t)}{dt} \]

\[ v(t) = \frac{dx(t)}{dt} \]

An ideal oscillator:

\[ m \frac{dv(t)}{dt} = -\frac{x(t) - x_0}{C} \]
Modeling a physical system

From first principles …

Hooke’s Law:

\[ F = -\frac{x - x_0}{C} \]

Newton’s Second:

\[ F = ma \]

A bit of calculus:

\[ a(t) = \frac{dv(t)}{dt} \]
\[ v(t) = \frac{dx(t)}{dt} \]

An ideal oscillator:

\[ m \frac{dv(t)}{dt} = -\frac{x(t) - x_0}{C} \]

Let’s develop a numerical solver to compute a solution …
**Euler:** step $h$ in time along $\dot{x} = f(x, t)$

$$\hat{x}_e(t_{k+1}) = x(t_k) + \dot{x}(t_k) h_k$$
Numerical integration

**Euler:** step $h$ in time along $\dot{x} = f(x, t)$

$$\hat{x}_e(t_{k+1}) = x(t_k) + \dot{x}(t_k)h_k$$
Numerical integration

*Euler*: step $h$ in time along $\dot{x} = f(x, t)$

$$\hat{x}_e(t_{k+1}) = x(t_k) + \dot{x}(t_k)h_k$$

*Trapezoidal*: average the end points

$$\hat{x}_t(t_{k+1}) = x(t_k) + \frac{\dot{x}(t_{k+1}) + \dot{x}(t_k)}{2}h_k$$
Numerical integration

**Euler:** step \( h \) in time along \( \dot{x} = f(x, t) \)

\[
\hat{x}_e(t_{k+1}) = x(t_k) + \dot{x}(t_k) h_k
\]

**Trapezoidal:** average the end points

\[
\hat{x}_t(t_{k+1}) = x(t_k) + \frac{\dot{x}(t_{k+1}) + \dot{x}(t_k)}{2} h_k
\]
**Numerical integration**

*Euler:* step $h$ in time along $\dot{x} = f(x, t)$

$$\hat{x}_e(t_{k+1}) = x(t_k) + \dot{x}(t_k) h_k$$

*Trapezoidal:* average the end points

$$\hat{x}_t(t_{k+1}) = x(t_k) + \frac{\dot{x}(t_{k+1}) + \dot{x}(t_k)}{2} h_k$$
Numerical integration

*Euler:* step \( h \) in time along \( \dot{x} = f(x, t) \)

\[
\hat{x}_e(t_{k+1}) = x(t_k) + \dot{x}(t_k)h_k
\]

*Trapezoidal:* average the end points

\[
\hat{x}_t(t_{k+1}) = x(t_k) + \frac{\dot{x}(t_{k+1}) + \dot{x}(t_k)}{2}h_k
\]

Taylor series expansion for error analysis

\[
x(t_{k+1}) = x(t_k) + \frac{\dot{x}(t_k)}{1!}h_k + \frac{\ddot{x}(t_k)}{2!}h_k^2 + O(h_k^3)
\]
Numerical integration

**Euler:** step $h$ in time along $\dot{x} = f(x, t)$

$$\hat{x}_e(t_{k+1}) = x(t_k) + \dot{x}(t_k) h_k$$

**Trapezoidal:** average the end points

$$\hat{x}_t(t_{k+1}) = x(t_k) + \frac{\dot{x}(t_{k+1}) + \dot{x}(t_k)}{2} h_k$$

Taylor series expansion for error analysis

$$x(t_{k+1}) = x(t_k) + \frac{\dot{x}(t_k)}{1!} h_k + \frac{\ddot{x}(t_k)}{2!} h_k^2 + O(h_k^3)$$

When $x(t)$ changes little, $h_k$ can be large!
Numerical integration

**Euler:** step $h$ in time along $\dot{x} = f(x, t)$

$$\hat{x}_e(t_{k+1}) = x(t_k) + \dot{x}(t_k) h_k$$

**Trapezoidal:** average the end points

$$\hat{x}_t(t_{k+1}) = x(t_k) + \frac{\dot{x}(t_{k+1}) + \dot{x}(t_k)}{2} h_k$$

Taylor series expansion for error analysis

$$x(t_{k+1}) = x(t_k) + \frac{\dot{x}(t_k)}{1!} h_k + \frac{\ddot{x}(t_k)}{2!} h_k^2 + \mathcal{O}(h_k^3)$$

$$\varepsilon_e(t_{k+1}) \quad \varepsilon_t(t_{k+1})$$

Change step size based on estimate:

$$\hat{x}_e(t_{k+1}) - \hat{x}_t(t_{k+1}) \approx \frac{\dot{x}(t_k)}{2!} h_k^2$$
Sophisticated solver … ?

- Let’s compute a solution to the ideal oscillator

- We can make the error small … but only locally!
Sophisticated solver … ?

- Let’s compute a solution to the ideal oscillator

- We can make the error small … but only locally!
- It accumulates for ‘long time’ behavior
- So, … how come the JSF flies?!
Engineering an embedded system

physical → theoretical → computational

validate → [] → verify

$$F(t) = m \frac{dv(t)}{dt}$$

```c
void main () {
    int i;
}
```
Engineering an embedded system

Physical → Theoretical → Computational

$F(t) = m \frac{dv(t)}{dt}$

void main () {
  int i;
}

In collaboration with Hans Vangheluwe, McGill University
Engineering an embedded system

physical → theoretical → computational

validate → [ ] → verify → void main () {
    int i;
}

refine → validate

In collaboration with Hans Vangheluwe, McGill University
Create executable models in all phases
Make the computational approximation the primary design deliverable—the real thing!
So that gets the job done … but …

- More than 50% of the modeling effort is in verification, validation, and testing!
- Semantics of models is buried in the execution engine
- Engine code base is extensive and complex
  - Interaction of approximations
  - Interaction and interfacing of different formalisms
- How can we mature the field?
- Model the semantics of the execution engine!
Agenda

- Outline
- Model-Based Design
- Problem statement
- A solution approach
- Outlook
So, what is a model anyway?

Jean Bézivin: “Everything is a model”

Jean-Marie Favre: “Nothing is a model”

Pieter J. Mosterman: “Nothing is not a model”

In collaboration with Hans Vangheluwe, McGill University
So, what is a model anyway?

Jean Bézivin

“Everything is a model”

Jean-Marie Favre

“Nothing is a model”

Pieter J. Mosterman

“Nothing is not a model”

Hans Vangheluwe

“Model everything”
So, what is a model anyway?

“Everything is a model”

Jean Bézivin

“Nothing is a model”

Jean-Marie Favre

“Nothing is not a model”

Pieter J. Mosterman

“Model everything”

Hans Vangheluwe

With the most appropriate formalism

At the most appropriate level of abstraction

In collaboration with Hans Vangheluwe, McGill University
Modeling the execution engine

analyze behavior

Host stack

Target stack
Modeling the execution engine
Create abstractions in the simulation stack ...

specification

implementation

Host stack
Create abstractions in the simulation stack …

declarative model

imperative model

implementation

Host stack
Analyze as little as possible

declarative model

Host stack
Further facilitate design, reuse, ...
A declarative formalism with fix-point semantics

- Repeated application of a monotonically increasing partial function converges to a fixed point

![Diagram](image)
A declarative formalism with fix-point semantics

- Repeated application of a monotonically increasing partial function converges to a fixed point
A declarative formalism with fix-point semantics

- Repeated application of a monotonically increasing partial function converges to a fixed point

- One implementation is a data dependency schedule
A declarative formalism with fix-point semantics

- Repeated application of a monotonically increasing partial function converges to a fixed point

- One implementation is a data dependency schedule
Dynamic systems evolve over time

- Sequences of fix-point evaluations
- Define input and output signals as (potentially infinite) streams of values
  - $$\text{Stream}(\text{Type}) = \text{Type} : \text{Stream}(\text{Type})$$
- Delay as a function application
  - $$\text{Delay } x_0 \ u = x_0 : u$$

\[
\begin{array}{c}
\left\{2, 4, 7, \ldots\right\} \\
\frac{1}{z} \\
\left\{5, 2, 4, \ldots\right\}
\end{array}
\]

- A variable has a ‘clock’ that encodes its sample time
Multiple rates; a potential problem …

- Streams are only practical if we can limit the stream entries being accessed
- Not this:
Clock calculus to detect

- Require compatible **clocks**: the synchronous assumption
- Match against base clock
Clock calculus to detect

- Require compatible **clocks**: the synchronous assumption
- Match against base clock
A multi-rate system example

Source: \{2, 7, 3, \ldots\}

Gain: 2

Delay: \frac{1}{z}

Delay: Ts = 1 (s)

Scope

Find greatest common divisor (Ts)!

Ts = 2 (s)
A multi-rate system example

Source

\{2, 7, 3, \ldots\}  \quad Ts = 2 \text{ (s)}

Gain

2

Delay

1/z

Scope

Base

Delay

Ts = 1 \text{ (s)}

Ts
A multi-rate system example

Source
{2, 7, 3, …}

Ts = 2 (s)

Gain
2

Delay
1/z

Scope

Delay
Ts = 1 (s)

Base
A multi-rate system example

Source: \{2, 7, 3, \ldots\}

Gain: 2

Delay: 1/z

Scope

Source

Delay

Base

Ts = 2 (s)

Ts = 1 (s)
A multi-rate system example

\{2, 7, 3, \ldots\} \quad Ts = 2 \ (s)

\begin{align*}
\text{Source} & \quad T & F & T & F & T & F & T & T \\
\text{RT} & \quad T & T & T & T & T & T & T & T \\
\text{Delay} & \quad T & T & T & T & T & T & T & T \\
\text{Base} & \quad T & T & T & T & T & T & T & T \\
\end{align*}
Can we use this framework to define a variable-step solver?

- Separate
  - Time (explicit)
  - Evaluations (ordered)

- Time as a function of evaluations
Can we use this framework to define a variable-step solver?

- Separate
  - Time (explicit)
  - Evaluations (ordered)
- Time as a function of evaluations
Can we use this framework to define a variable-step solver?

- Separate
  - Time (explicit)
  - Evaluations (ordered)

- Time as a function of evaluations
Can we use this framework to define a variable-step solver?

- Separate
  - Time (explicit)
  - Evaluations (ordered)
- Time as a function of evaluations
Can we use this framework to define a variable-step solver?

- Separate
  - Time (explicit)
  - Evaluations (ordered)
- Time as a function of evaluations
Can we use this framework to define a variable-step solver?

- **Separate**
  - Time (explicit)
  - Evaluations (ordered)

- **Time as a function of evaluations**
  - Step is variable
Can we use this framework to define a variable-step solver?

- Separate
  - Time (explicit)
  - Evaluations (ordered)

- Time as a function of evaluations
  - Step is variable
  - Step may be 0
Can we use this framework to define a variable-step solver?

- Separate
  - Time (explicit)
  - Evaluations (ordered)

- Time as a function of evaluations
  - Step is variable
  - Step may be 0
  - Step may be negative
    - Time may recede
Can we use this framework to define a variable-step solver?

- Separate
  - Time (explicit)
  - Evaluations (ordered)

- Time as a function of evaluations
  - Step is variable
  - Step may be 0
  - Step may be negative
    - Time may recede
A stream based functional solver

**Euler integration**

\[ y_e(e) = \begin{cases} \sum_{i=1}^{e} u(i)h(i) & \text{if } \text{odd}(e) \\ y_e(e-1) & \text{otherwise} \end{cases} \]

**Trapezoidal integration**

\[ y_t(e) = \sum_{i=1}^{e} \frac{(u(i-1) + u(i))h(i-1)}{2} \]
A stream based functional solver

Euler integration

\[ y_e(e) = \begin{cases} 
\sum_{i=1}^{e} u(i)h(i) - u(i - 2)h(i - 2)p(i) & \text{if } \text{odd}(e) \\
y_e(e - 1) & \text{otherwise}
\end{cases} \]

Trapezoidal integration

\[ y_t(e) = \sum_{i=1}^{e} \frac{(u(i - 1) + u(i))h(i - 1)}{2} + \frac{(u(i - 3) + u(i - 2))h(i - 3)}{2}p(i - 1) \]

Error computation

\[ d(e) = \frac{(u(e - 3) + u(e - 2))h(e - 3)}{2} - u(e - 2)h(e - 2) < \text{tol} \]
Now we can create a variable step solver inside $1/s$ that maps onto the synchronous paradigm.

Dynamically compute ‘hold’ output as an argument of time ...

$$y(t) = u(t / T_s)$$

No predetermined sequence of output values.
Unifying formalisms with different semantics

- Newton’s Law and Hooke’s Law
  - Differential equations as before

- Control behavior
  - Sampled data (periodic $T_s=0.5$)
    \[ F_{pull}(k) = \begin{cases} 
    20 & \text{if } k = 0 \\
    10 & \text{if } k = 1 \\
    0 & \text{else} 
    \end{cases} \]

- Contact behavior
  - Discontinuous changes …
Modeling the contact behavior

- Simultaneous inequalities

\[ F_{floor}(t) = \begin{cases} -\left( Rv(t) + \frac{x(t)}{C} \right) & \text{if } x(t) < 0 \\ 0 & \text{otherwise} \end{cases} \]

- Finite state machine

\[ F_{floor}(t) = 0 \] (free) \quad \text{if } x(t) < 0 \quad \text{and} \quad F_{floor}(t) = -\left( Rv(t) + \frac{x(t)}{C} \right) \] (contact) \quad \text{if } x(t) \geq 0
Computational simulation

Position vs. time

Simultaneous inequalities

<table>
<thead>
<tr>
<th>Eval</th>
<th>Time</th>
<th>Position</th>
<th>Velocity</th>
<th>$F_{\text{floor}}$</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>532</td>
<td>2.5450</td>
<td>0.0037</td>
<td>-1.4381</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>533</td>
<td>2.5450</td>
<td>-0.0035</td>
<td>-1.4381</td>
<td>8.5888</td>
<td>0</td>
</tr>
<tr>
<td>534</td>
<td>2.5550</td>
<td>-0.0107</td>
<td>-1.4398</td>
<td>11.4717</td>
<td></td>
</tr>
<tr>
<td>535</td>
<td>2.5550</td>
<td>-0.0179</td>
<td>-1.4377</td>
<td>14.3430</td>
<td>0.0021</td>
</tr>
<tr>
<td>536</td>
<td>2.5500</td>
<td>-0.0035</td>
<td>-1.4348</td>
<td>8.5700</td>
<td></td>
</tr>
<tr>
<td>537</td>
<td>2.5500</td>
<td>-0.0107</td>
<td>-1.4369</td>
<td>11.4555</td>
<td>0.0021</td>
</tr>
<tr>
<td>538</td>
<td>2.5475</td>
<td>0.0001</td>
<td>-1.4375</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>539</td>
<td>2.5475</td>
<td>-0.0071</td>
<td>-1.4395</td>
<td>10.0333</td>
<td>0.0020</td>
</tr>
<tr>
<td>540</td>
<td>2.5462</td>
<td>0.0019</td>
<td>-1.4380</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>541</td>
<td>2.5462</td>
<td>-0.0053</td>
<td>-1.4389</td>
<td>9.3125</td>
<td>0.0009</td>
</tr>
<tr>
<td>542</td>
<td>2.5456</td>
<td>0.0028</td>
<td>-1.4381</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>543</td>
<td>2.5456</td>
<td>-0.0044</td>
<td>-1.4385</td>
<td>8.9508</td>
<td>0.0004</td>
</tr>
</tbody>
</table>
Computational simulation

Position vs. time

Simultaneous inequalities

<table>
<thead>
<tr>
<th>Eval</th>
<th>Time</th>
<th>Position</th>
<th>Velocity</th>
<th>$F_{floor}$</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>532</td>
<td>2.5450</td>
<td>-0.0037</td>
<td>-1.4381</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>533</td>
<td>2.5450</td>
<td>-0.0035</td>
<td>-1.4381</td>
<td>8.5888</td>
<td>0</td>
</tr>
<tr>
<td>534</td>
<td>2.5550</td>
<td>-0.0107</td>
<td>-1.4398</td>
<td>11.4717</td>
<td>0</td>
</tr>
<tr>
<td>535</td>
<td>2.5550</td>
<td>-0.0179</td>
<td>-1.4377</td>
<td>14.3430</td>
<td>0.0021</td>
</tr>
<tr>
<td>536</td>
<td>2.5500</td>
<td>-0.0035</td>
<td>-1.4348</td>
<td>8.5700</td>
<td>0</td>
</tr>
<tr>
<td>537</td>
<td>2.5500</td>
<td>-0.0107</td>
<td>-1.4369</td>
<td>11.4555</td>
<td>0.0021</td>
</tr>
<tr>
<td>538</td>
<td>2.5475</td>
<td>0.0001</td>
<td>-1.4375</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>539</td>
<td>2.5475</td>
<td>-0.0071</td>
<td>-1.4395</td>
<td>10.0333</td>
<td>0.0020</td>
</tr>
<tr>
<td>540</td>
<td>2.5462</td>
<td>0.0019</td>
<td>-1.4380</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>541</td>
<td>2.5462</td>
<td>-0.0053</td>
<td>-1.4389</td>
<td>9.3125</td>
<td>0.0009</td>
</tr>
<tr>
<td>542</td>
<td>2.5456</td>
<td>0.0028</td>
<td>-1.4381</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>543</td>
<td>2.5456</td>
<td>-0.0044</td>
<td>-1.4385</td>
<td>8.9508</td>
<td>0.0004</td>
</tr>
</tbody>
</table>

Time vs. evaluations (detail)

Finite state machine

<table>
<thead>
<tr>
<th>Eval</th>
<th>Time</th>
<th>Position</th>
<th>Velocity</th>
<th>$F_{floor}$</th>
<th>Error</th>
<th>$\xi_{con}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>532</td>
<td>2.5450</td>
<td>-0.0037</td>
<td>-1.4381</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>533</td>
<td>2.5450</td>
<td>-0.0035</td>
<td>-1.4381</td>
<td>8.5888</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>534</td>
<td>2.5550</td>
<td>-0.0107</td>
<td>-1.4398</td>
<td>11.4717</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>535</td>
<td>2.5550</td>
<td>-0.0179</td>
<td>-1.4377</td>
<td>14.3430</td>
<td>0.0021</td>
<td></td>
</tr>
<tr>
<td>536</td>
<td>2.5500</td>
<td>-0.0035</td>
<td>-1.4348</td>
<td>8.5700</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>537</td>
<td>2.5500</td>
<td>-0.0107</td>
<td>-1.4369</td>
<td>11.4555</td>
<td>0.0021</td>
<td></td>
</tr>
<tr>
<td>538</td>
<td>2.5475</td>
<td>0.0001</td>
<td>-1.4375</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>539</td>
<td>2.5475</td>
<td>-0.0071</td>
<td>-1.4395</td>
<td>10.0333</td>
<td>0.0020</td>
<td></td>
</tr>
<tr>
<td>540</td>
<td>2.5462</td>
<td>0.0019</td>
<td>-1.4380</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>541</td>
<td>2.5462</td>
<td>-0.0053</td>
<td>-1.4389</td>
<td>9.3125</td>
<td>0.0009</td>
<td></td>
</tr>
<tr>
<td>542</td>
<td>2.5456</td>
<td>0.0028</td>
<td>-1.4381</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>543</td>
<td>2.5456</td>
<td>-0.0044</td>
<td>-1.4385</td>
<td>8.9508</td>
<td>0.0004</td>
<td></td>
</tr>
</tbody>
</table>
Computational simulation

Position vs. time

Time vs. evaluations (detail)

Simultaneous inequalities

Finite state machine
But time is a function of evaluations

- Simultaneous inequalities

\[
F_{\text{floor}}(t_{\text{event}}(e)) = \begin{cases} 
- \left( Rv(t_{\text{event}}(e)) + \frac{x(t_{\text{event}}(e))}{C} \right) & \text{if } x(t_{\text{event}}(e)) < 0 \\
0 & \text{otherwise}
\end{cases}
\]

\[
a_{\text{ball}}(t_{\text{smooth}}(e)) = g + \frac{F_{\text{floor}}(t_{\text{smooth}}(e))}{m_{\text{ball}}}
\]

- Which \( t \) should \( t_{\text{event}} \) really map onto?
Different choice of semantics

always evaluated

evaluated on accepted time step
Comparing with an analytic solution

Characteristics of the semantic domain

- Declarative
  - Purely functional (no side effects)
- Ordered evaluations
- Untimed
  - Time as explicit function, $t(e)$
  - Time is not strictly increasing
- Broadly applicable to dynamic systems
  - Differential equations, difference equations, discrete events

Agenda

- Outline
- Model-Based Design
- Problem statement
- A solution approach
- Outlook
Conclusions

- Computation, the good and not so good
  - Quantitative approximation
  - Potential for higher quality models
- Exploit computational methods
  - We must formalize the computational execution semantics
  - Model at a declarative level
- Define solvers using a functional stream-based approach
  - Precise computational semantics of the execution engine
Opportunities

- First principles in computational form
- New generation of modeling and simulation tools
  - More robust in less time (less bugs, more reliable and consistent approximation)
- Bring disciplines together
  - Engineering, Computer Science, Physics, Mathematics
- Exploit the abstraction
  - Automatic code generation
  - Computational methods for
    - Analysis
    - Design
    - Synthesis
Control synthesis using model checking
Control synthesis using model checking
A counterexample

A counterexample

Controlled acceleration of mounted component
Acknowledgments

Justyna Zander
Harvard University
Fraunhofer Institute FOKUS, Berlin

Grégoire Hamon
MathWorks

Ben Denckla
Independent Thinker

Hans Vangheluwe
University of Antwerp
McGill University

Many thanks for their continuing collaboration!
MathWorks®

Accelerating the pace of engineering and science