

ROBUSTNESS OF MODEL-BASED SIMULATIONS

Georgios Fainekos, Arizona State University♣

Sriram Sankaranarayanan, University of Colorado♣

Franjo Ivancic, NEC Labs

Aarti Gupta, NEC Labs

♣ Work performed
at NEC Labs

RTSS 2009, Washington D.C., 2009.12.04

Round-off and truncation errors

Consider the function (Rump's example)*

$$f(a, b) = (333.75 - a^2)b^6 + a^2(11a^2b^2 - 121b^4 - 2) + 5.5b^8 + a/2b$$

Using round-to-nearest IEEE-754 arithmetic, the function f for $a = 77617$ and $b = 33096$ evaluates to

1.172604 (32-bit)

1.1726039400531786 (64-bit)

1.1726039400531786318588349045201838 (128-bit)

By increasing the precision, we seem to derive more accurate results!

*E. Loh and G. W. Walster. Rump's example revisited. *Reliable Computing*, 8:245248, 2002.

Round-off and truncation errors

3

But, this is deceiving!

The correct answer is:

$$f(a, b) = -0.827396059946821368141165095479816\dots$$

Not even the sign is correct!

The Patriot missile defense system failure*

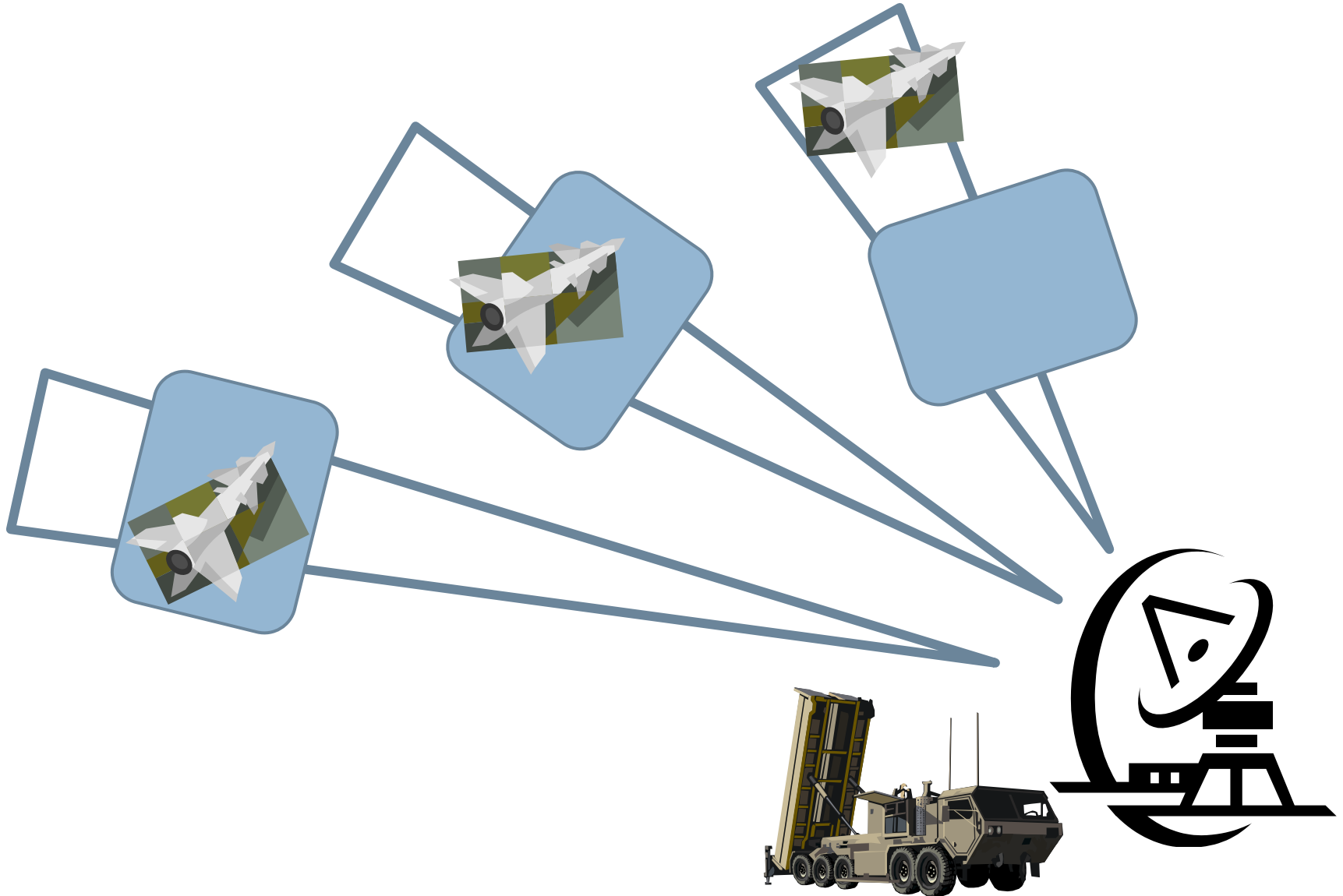
4

- The software estimated the future position of the incoming missile based on the velocity and the last position as returned by the radar
- The software modeled time in increments of *0.1 sec.*
- *0.1* itself can not be accurately represented in the underlying 24-bit fixed-point register.
- As a result, the accumulated errors caused a drift in the computed times when the system was operational for many hours.
- Ultimately, this led to a failure to track and intercept incoming missiles.

* M. Blair, S. Obenski, and P. Bridickas. Patriot missile software problem. Technical Report GAO/IMTEC-92-26, United States General Accounting Office, 1992

The Patriot missile defense system failure

5



In the analysis of Cyber-Physical Systems ...

6

- floating-point rounding and truncation errors are important
- but there exist more sources of errors
 - ▣ uncertain parameters
 - ▣ uncertain inputs
 - ▣ numerical integration errors

Contributions: What is this paper about?

7

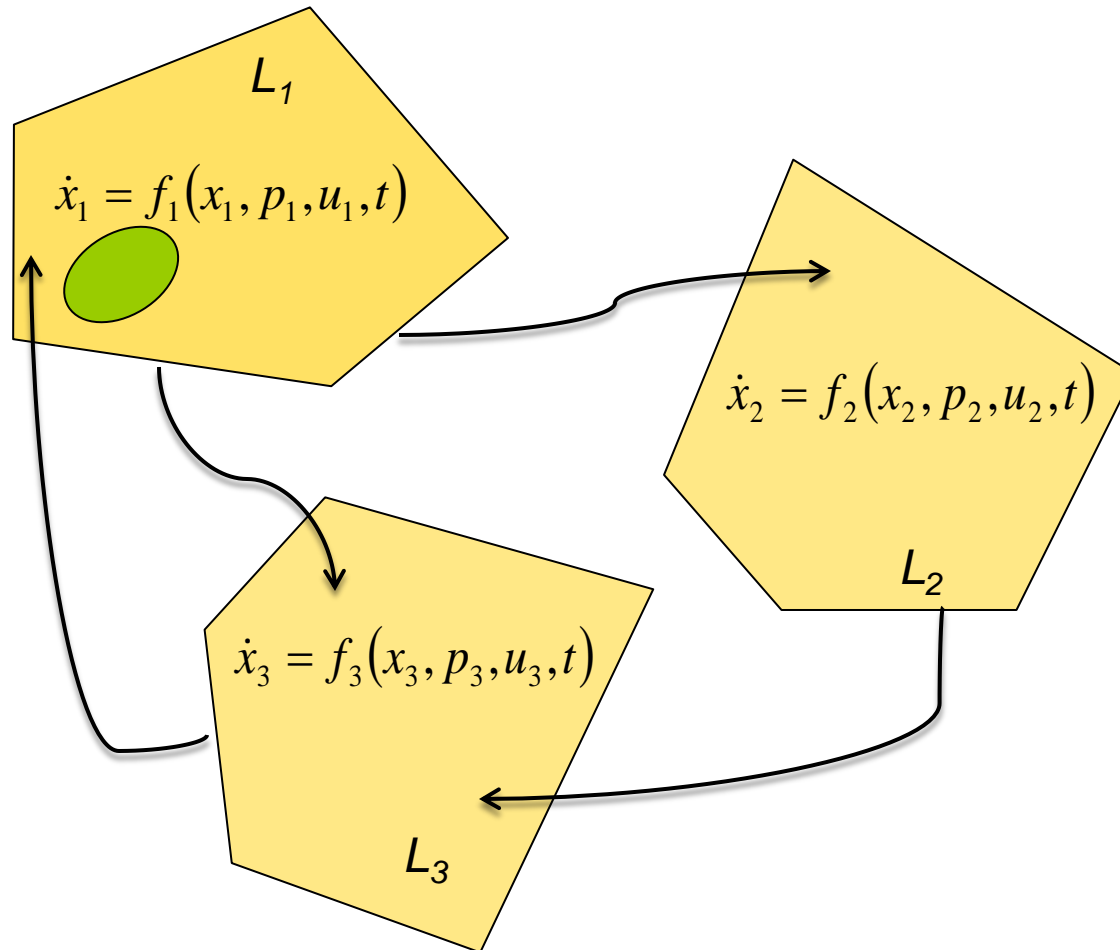
- We investigate the **robustness** of model generated simulations under
 - ▣ **numerical errors**
 - ▣ **floating-point rounding** and **truncation errors**
 - ▣ **parameter uncertainties**
 - ▣ **input uncertainties**
- The model based design environment we target is **Simulink**
 - ▣ **Validated simulation without model conversion**

The roadmap of this presentation

- ☑ Introduction
- ☐ The model of CPS that we use
- ☐ Our definition of robustness
- ☐ Formal problem definition
- ☐ Solution overview
- ☐ Validated arithmetics
- ☐ Details for discrete-time systems
- ☐ Details for continuous-time systems
- ☐ Details for mixed signal systems
- ☐ Conclusions

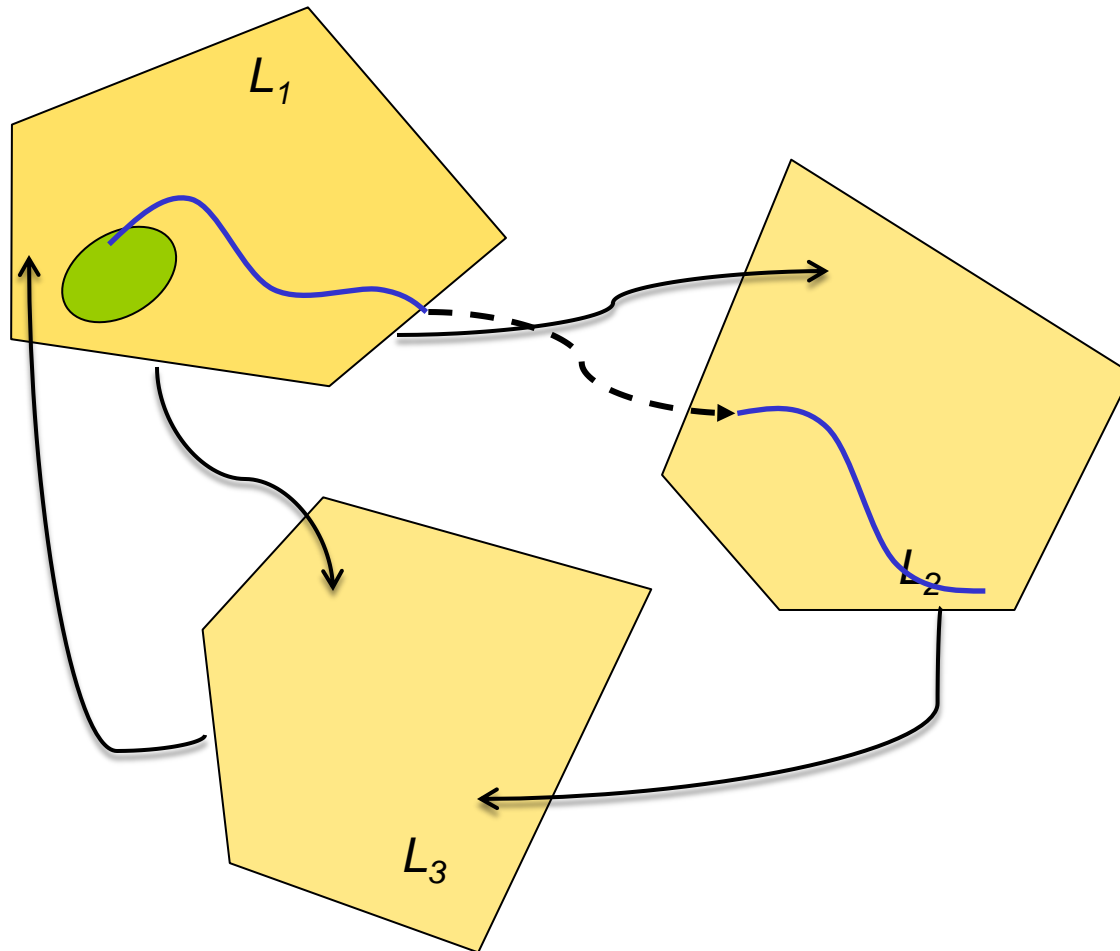
What is the model of CPS that we use?

9



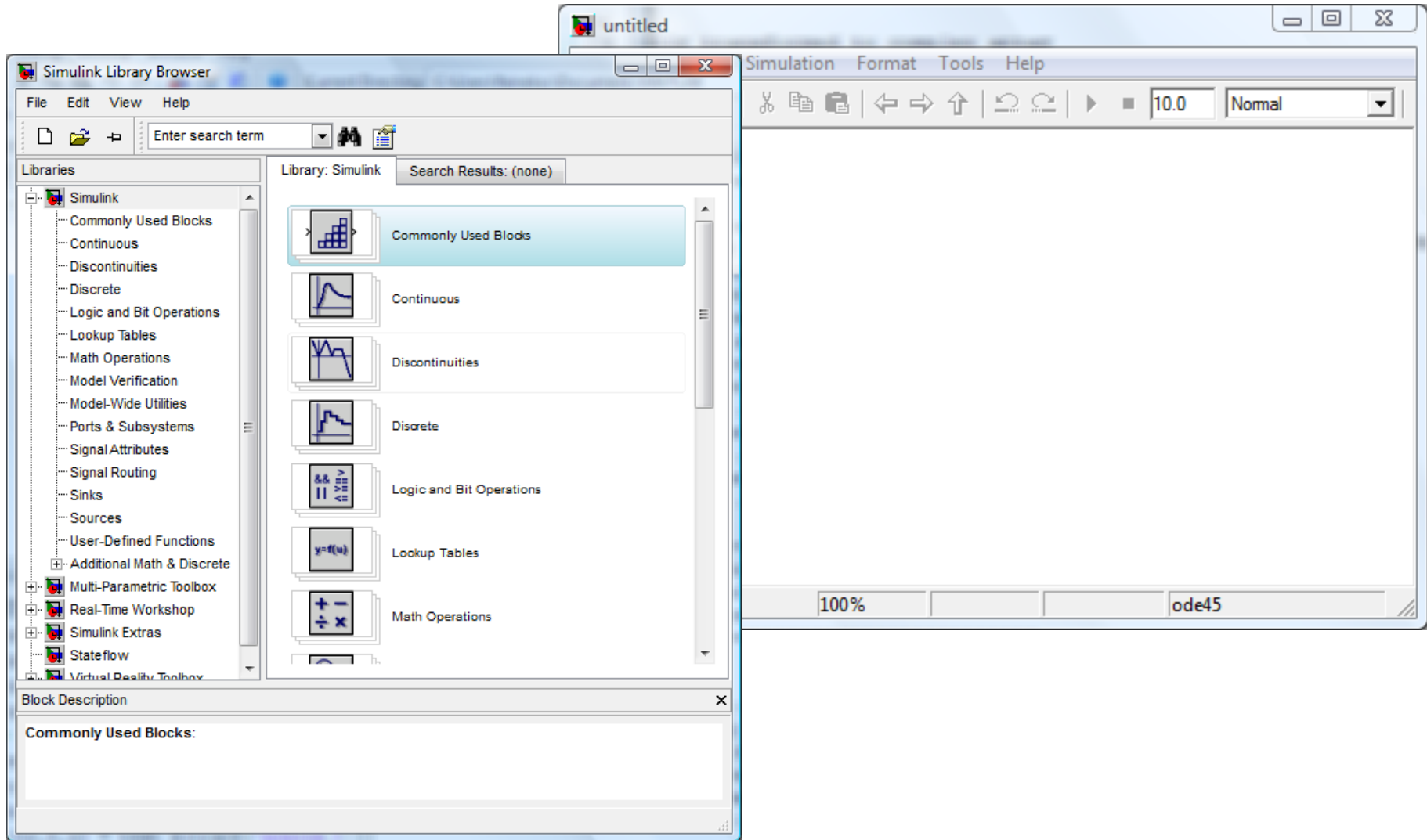
What is the model of CPS that we use?

10



Simulink

11



Simulink

12

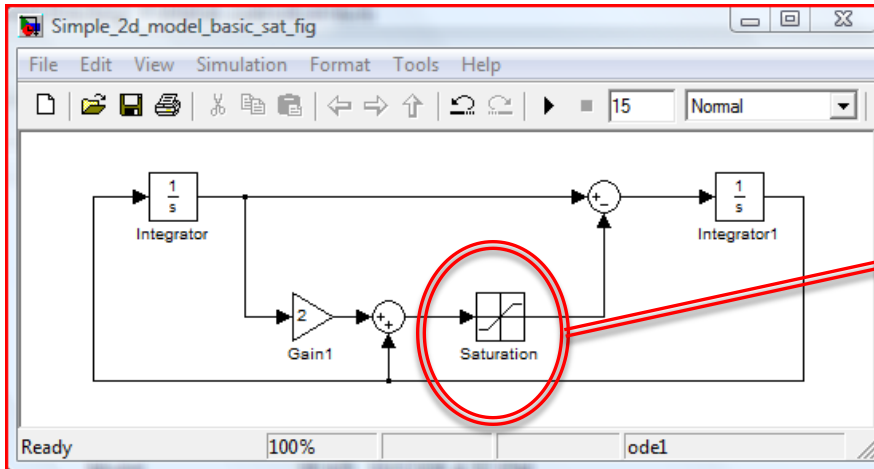
The image displays the Simulink software interface. On the left, the **Simulink Library Browser** is open, showing a tree view of libraries. The **Commonly Used Blocks** section is expanded, and the **Commonly Used Blocks** library is selected. The main workspace shows a Simulink model titled **Simple_2d_model_basic_sat_fig**. The model consists of the following blocks and connections:

- An input signal enters from the left and splits into two paths.
- The upper path goes through an **Integrator** block (represented by $\frac{1}{s}$).
- The lower path goes through a **Gain1** block (represented by the number 2).
- The outputs of the **Integrator** and **Gain1** blocks are summed at a junction.
- The output of this junction goes through a **Saturation** block.
- The output of the **Saturation** block is summed with the output of the **Integrator** block at a second junction.
- The output of this second junction goes through an **Integrator1** block (represented by $\frac{1}{s}$).
- The output of **Integrator1** is fed back to the input of the **Integrator** block, forming a closed-loop system.

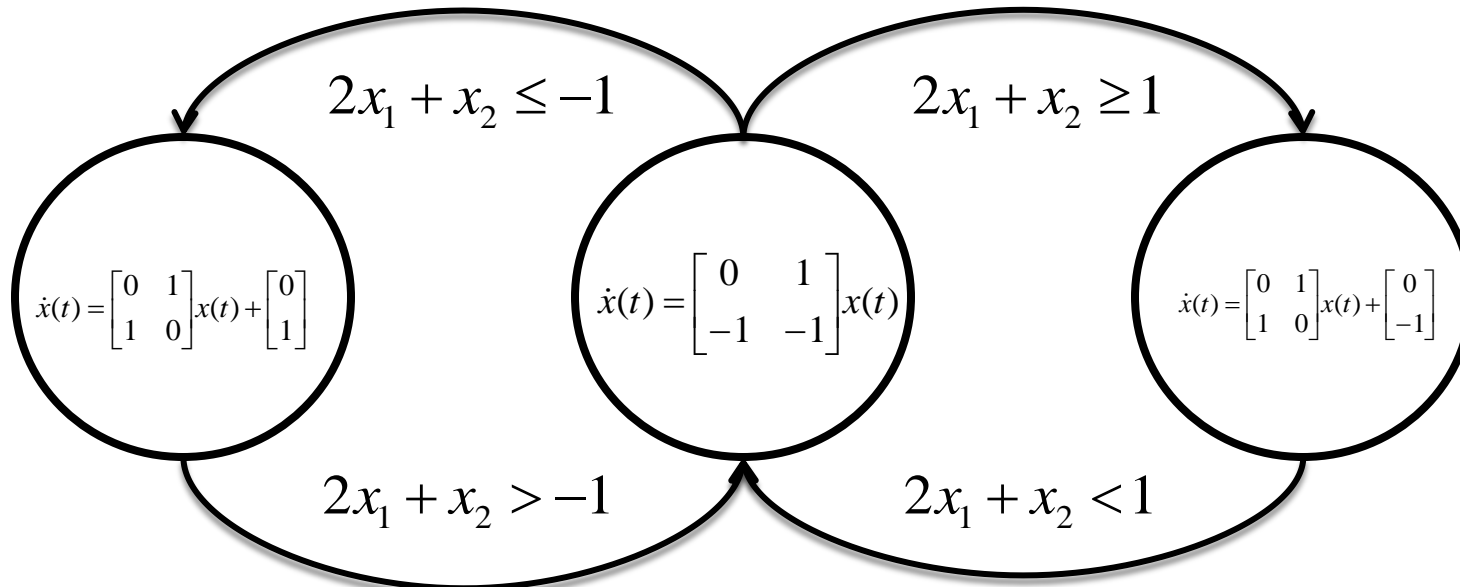
The status bar at the bottom of the model window shows **Ready**, **100%** zoom, and the solver type **ode1**.

Simulink & Hybrid Automata

13



```
if in_signal >= 1
    return 1
elseif in_signal <= -1
    return -1
else
    return in_signal
end
```



Our approach

14

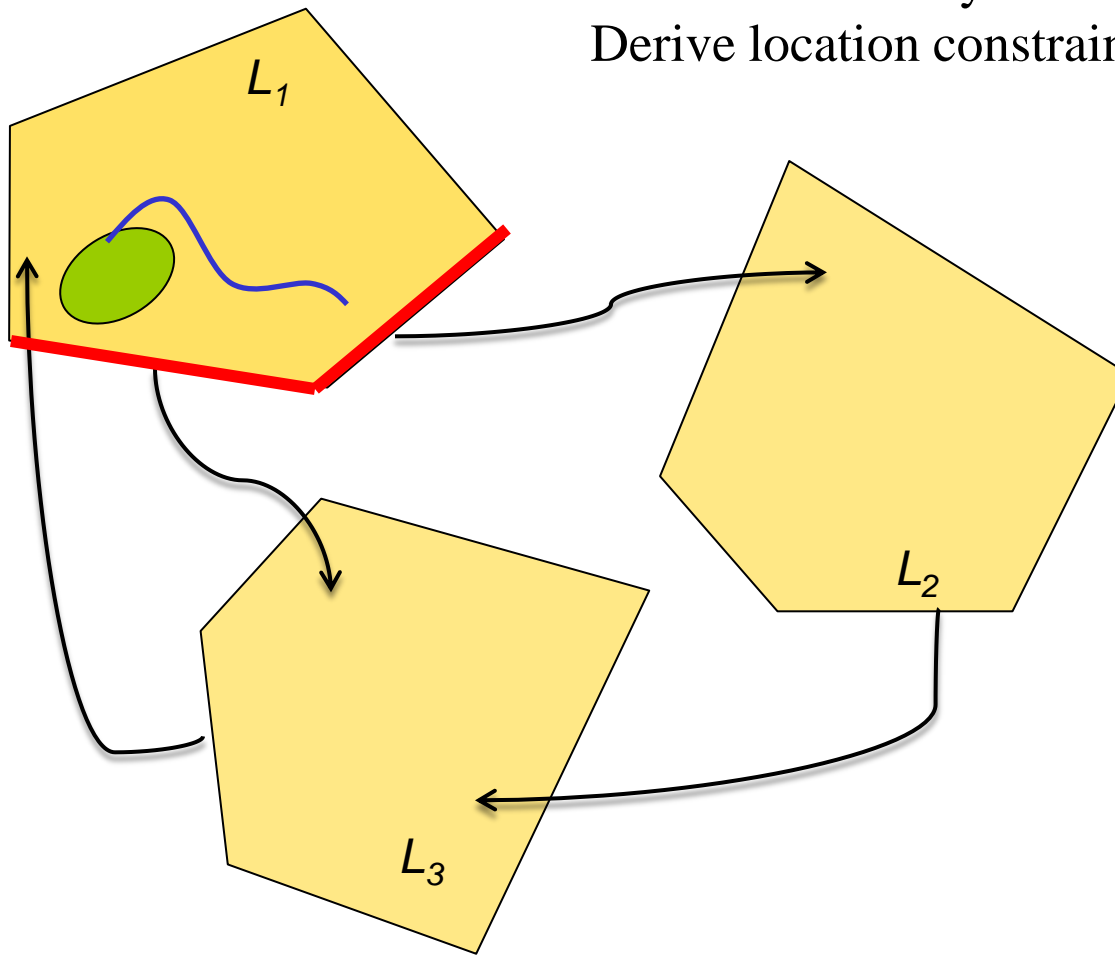
- We do not do such a translation
 - ▣ Potential exponential blow-up in the number of discrete locations
 - ▣ Unknown Simulink semantics

- Instead we simulate and capture the current operating point
 - ▣ We derive system dynamics
 - ▣ We derive system constraints

Concrete + Symbolic simulation

15

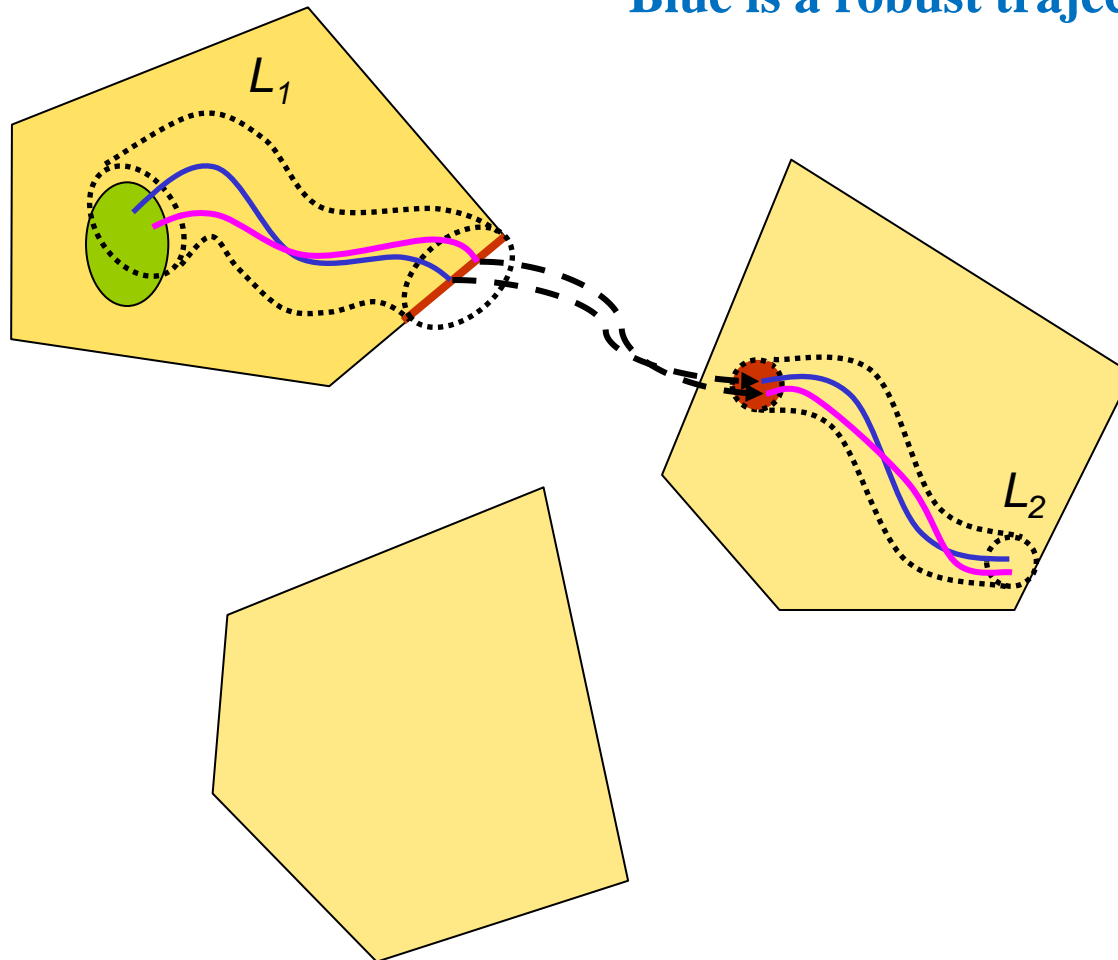
Do symbolic analysis of the Simulink model
Derive location dynamics: $dx_1/dt = f_1(x_1, p_1, u_1, t)$
Derive location constraints: $g(x_1, p_1, u_1, t) \leq 0$



What do we mean by robustness?

16

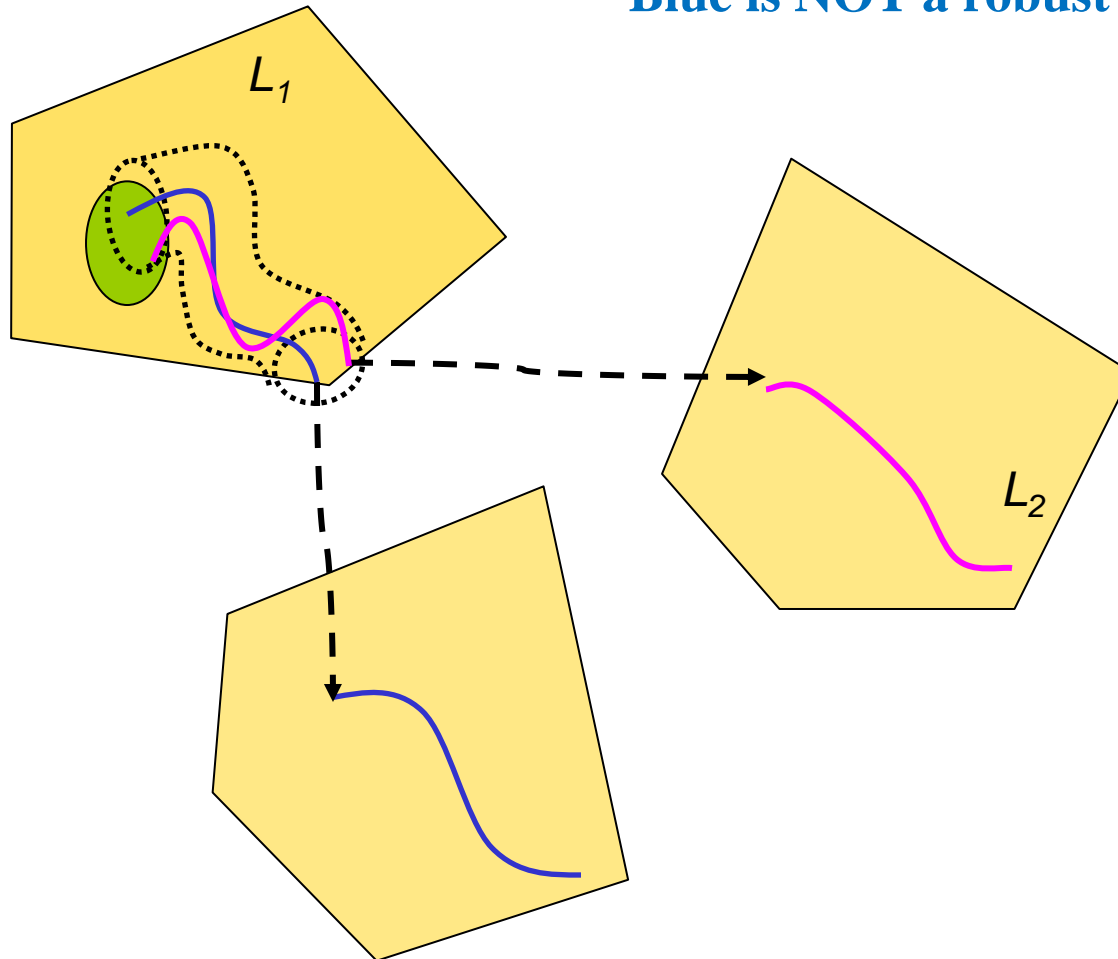
Blue is a robust trajectory



What do we mean by robustness?

17

Blue is NOT a robust trajectory



Trajectory robustness

18

- Under
 - ▣ Floating point errors
 - ▣ Uncertain parameters
 - ▣ Uncertain inputs
 - ▣ Possible numerical errors
- the trajectory of the system should follow the same sequence of discrete transitions as the simulated one

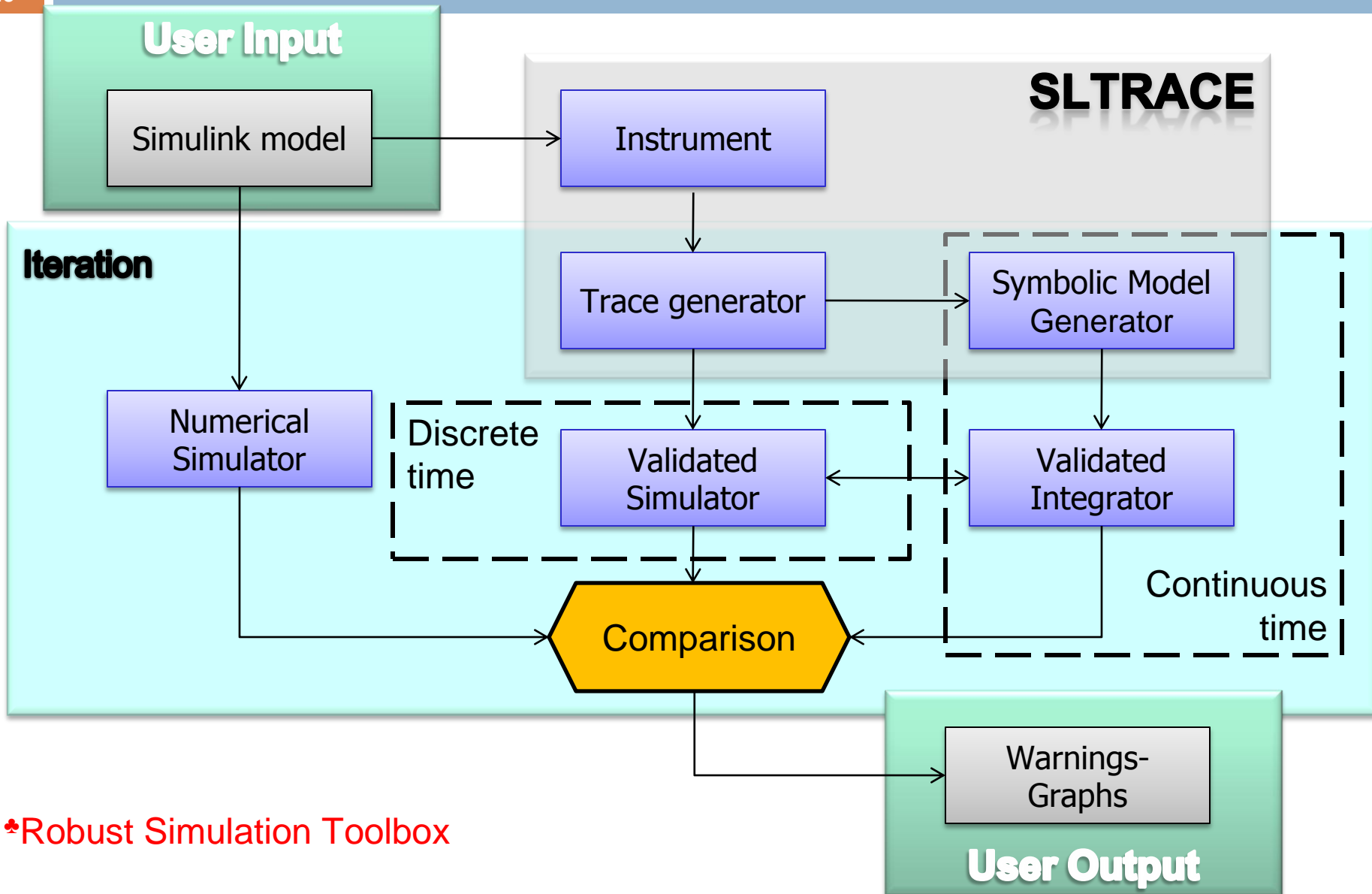
Problem definition

19

- Given a Simulink model S , a set of uncertain initial conditions X_0 and parameters P , determine points in time when the Simulink simulation trajectory is not robust.

Solution overview: RobSim♣

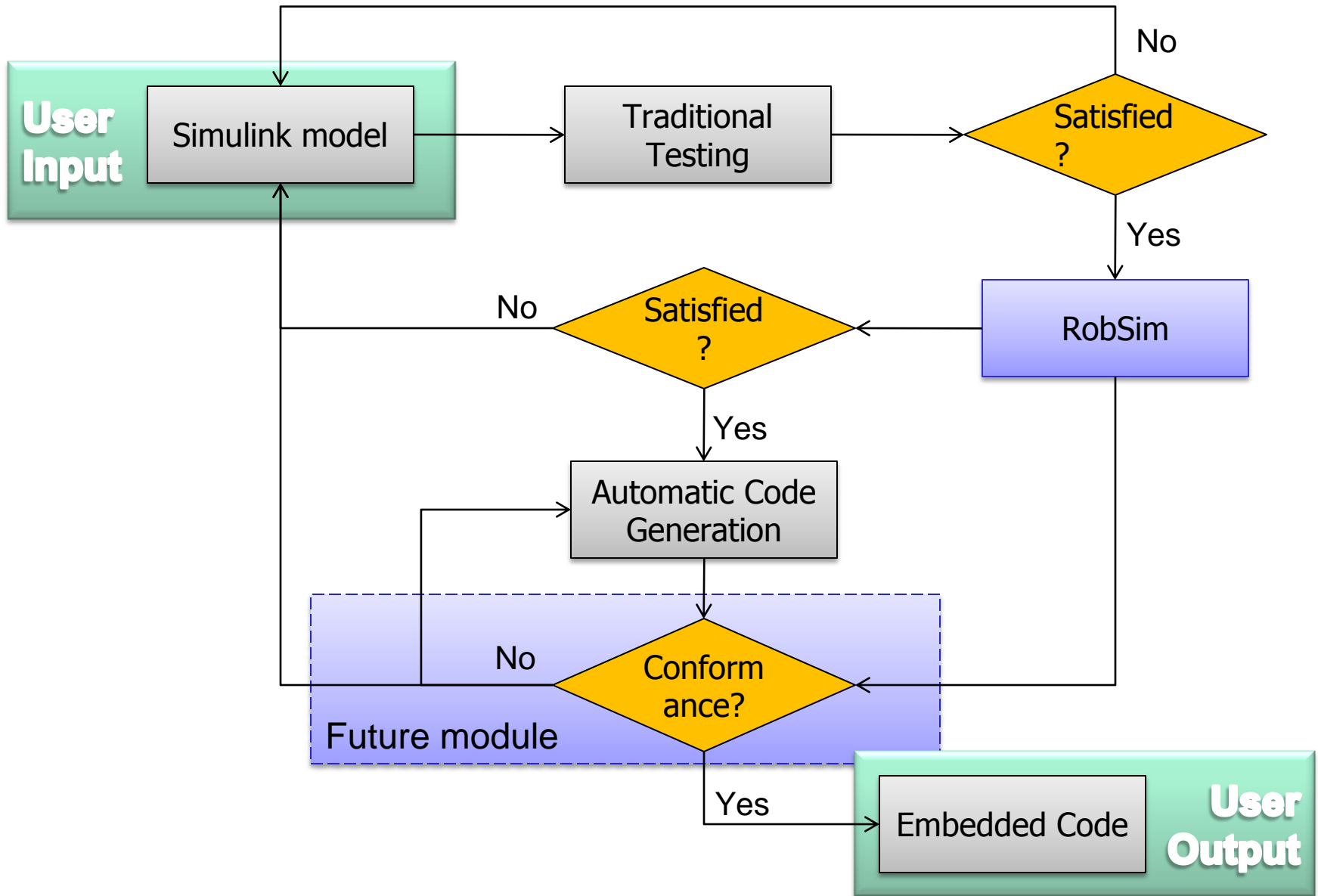
20



♣Robust Simulation Toolbox

Design Flow Using RobSim

21



Interval Arithmetic

22

- ❖ IA is an arithmetic for validated computations
 - introduced by Moore several decades ago
- ❖ IA computes a range in machine arithmetic that contains the result of the operation in real arithmetic
- ❖ An IA quantity x is a range $[x_l, x_u]$
- ❖ You can bound any nonlinear function
 - The approximation might be too conservative

Interval Arithmetic

23

- Addition:

$$x + y = [x_l, x_u] + [y_l, y_u] = [x_l + y_l, x_u + y_u]$$

- Multiplication:

$$\begin{aligned} xy &= [x_l, x_u][y_l, y_u] = \\ &= [\min\{x_ly_l, x_ly_u, x_uy_l, x_uy_u\}, \max\{x_ly_l, x_ly_u, x_uy_l, x_uy_u\}] \end{aligned}$$

- We can capture floating-point rounding errors by rounding down for the lower bound and rounding up for the upper bound
- Problems:
 - sub-distributive, i.e., $[x] ([y] + [z]) \subseteq [x] [y] + [x] [z]$
 - it has no additive and no multiplicative inverse.

Affine Arithmetic

24

- AA is a tool for validated computations
 - ▣ introduced by Comba and Stolfi in 1993
- AA keeps track of *first-order correlations* in computations
 - ▣ AA provides tighter interval estimates than IA in many cases
 - ▣ AA provides additional information that can be exploited
- AA represents a quantity x with an *affine form*

$$\langle x \rangle = x_0 + x_1 \varepsilon_1 + x_2 \varepsilon_2 + \dots + x_n \varepsilon_n$$

- ▣ $x_0 \in \mathfrak{R}$ is the central value
- ▣ $\varepsilon_i \in [-1, +1]$ are independent but unknown terms
- ▣ $x_i \in \mathfrak{R}$ are coefficients assigning a weight to each term
- ▣ n is *not fixed*, new terms are created during computation
- ▣ If $\alpha \in \langle x \rangle$, then $\alpha \in [x_0 - |x_1| - \dots - |x_n|, x_0 + |x_1| + \dots + |x_n|]$

Affine Arithmetic

25

□ Addition

$$\blacksquare \langle x \rangle + \langle y \rangle = (x_0 + \sum_i x_i \varepsilon_i) + (y_0 + \sum_i y_i \varepsilon_i) = (x_0 + y_0) + \sum (x_i + y_i) \varepsilon_i$$

□ Multiplication

$$\blacksquare \langle x \rangle \langle y \rangle = (x_0 + \sum_i x_i \varepsilon_i)(y_0 + \sum_i y_i \varepsilon_i) = x_0 y_0 + \sum (x_0 y_i) \varepsilon_i + \sum (x_i y_0) \varepsilon_i + z \varepsilon_{k+1}$$

□ Any nonlinear function can be approximated

▣ The approximation can be conservative

□ The usual properties of real arithmetic hold

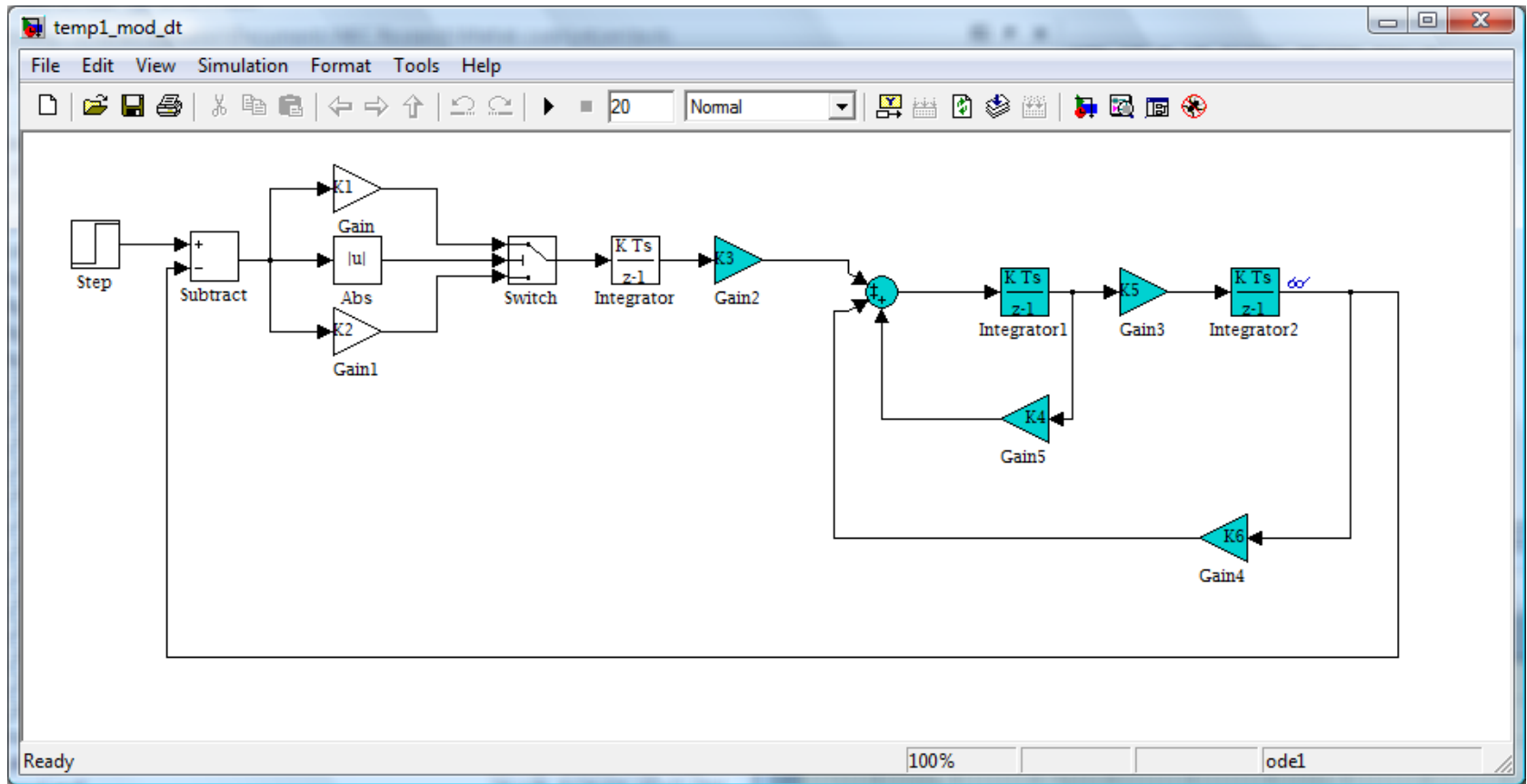
□ We created our own toolbox for affine arithmetic in Matlab

▣ AffLab

▣ which is based on IntLab the interval arithmetic toolbox by Rump.

A discrete time example

26

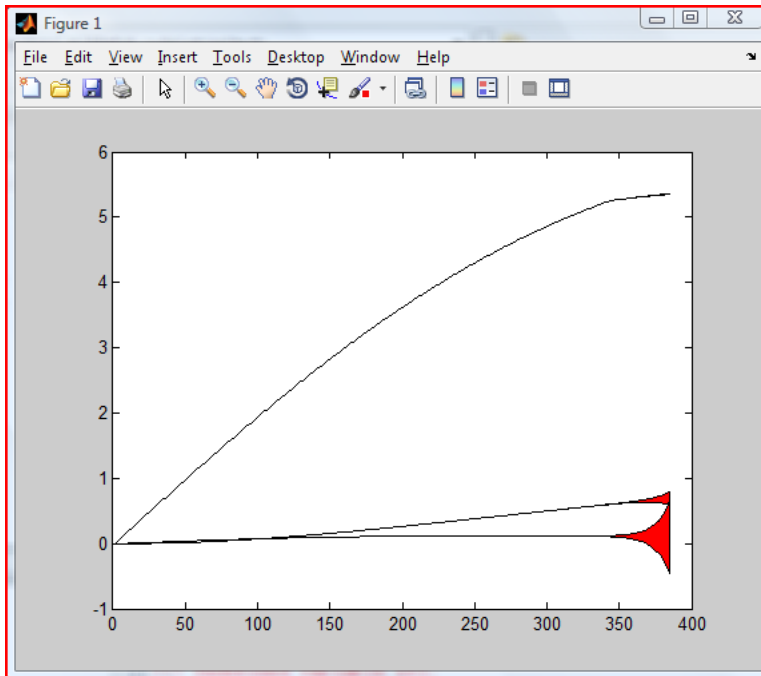


Comparison of the 2 arithmetics

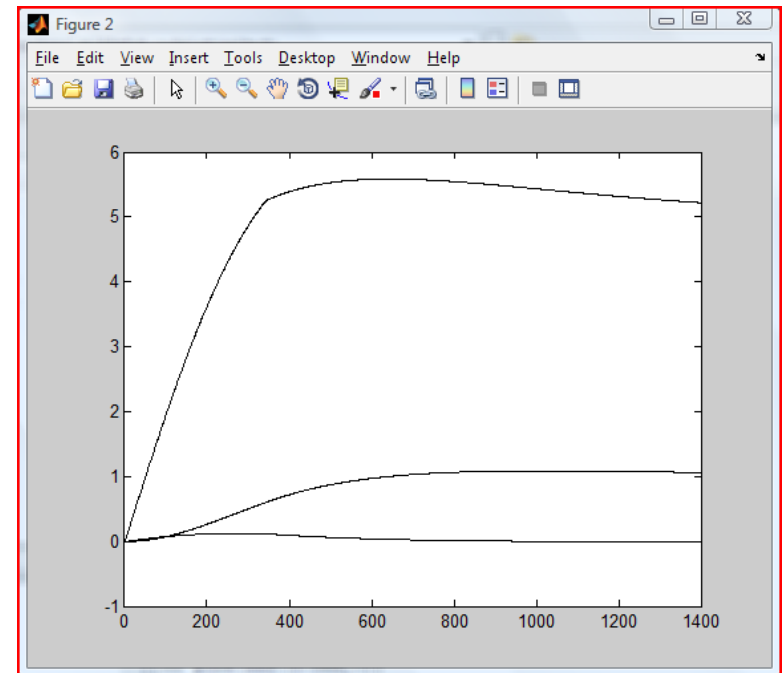
27

Computation time for 1400 samples \approx 17sec

Computation time for 1400 samples \approx 122sec,
number of affine terms 11187



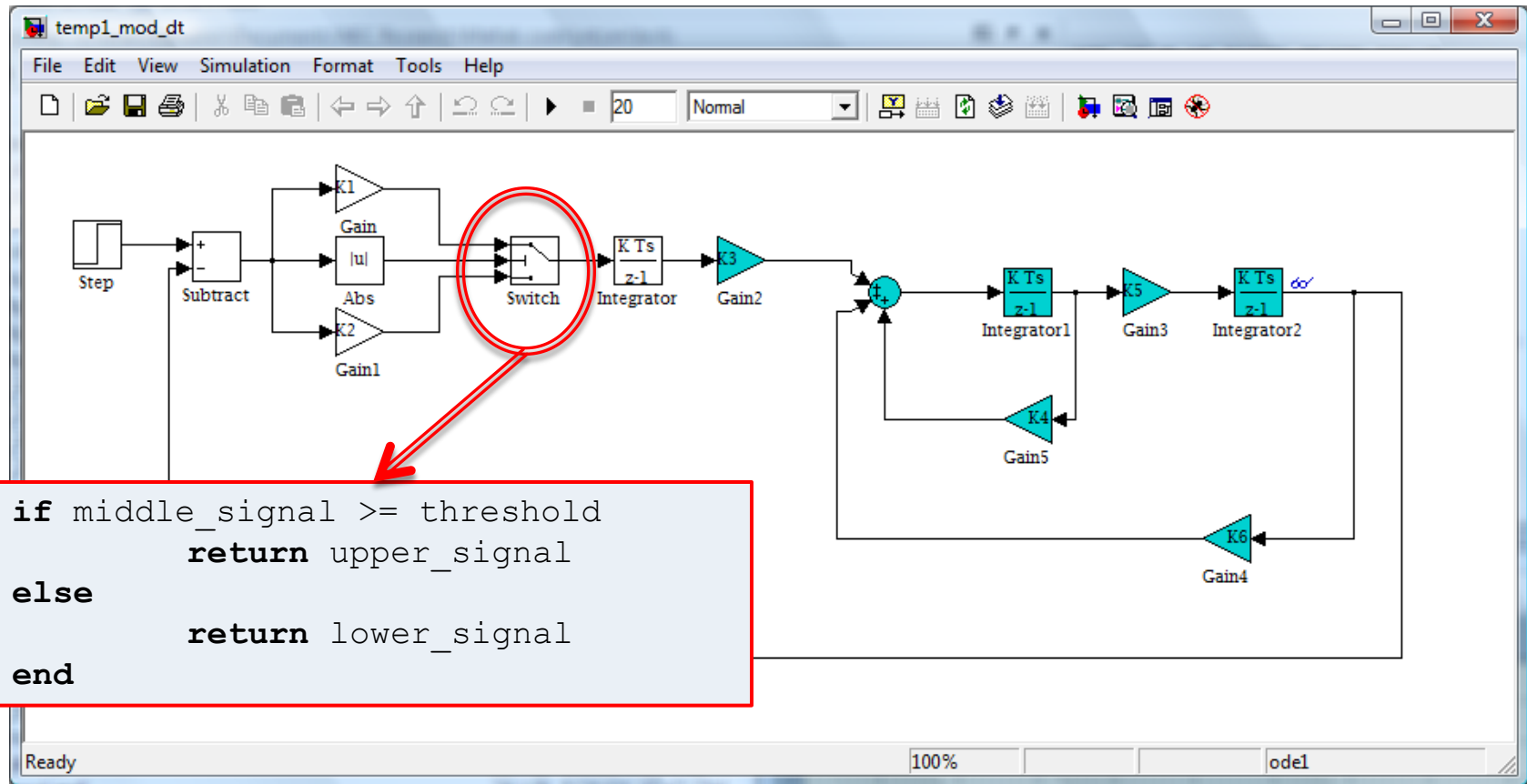
(a) Result of the simulation using interval arithmetic. ValSim warns that at sampling point 342 the *Switch* could have chosen either signal due to accumulated errors.



(b) The same simulation using affine arithmetic. The result indicates that the Simulink simulation is correct. The warning in (a) is due to conservative approximations of the rounding errors by interval arithmetic.

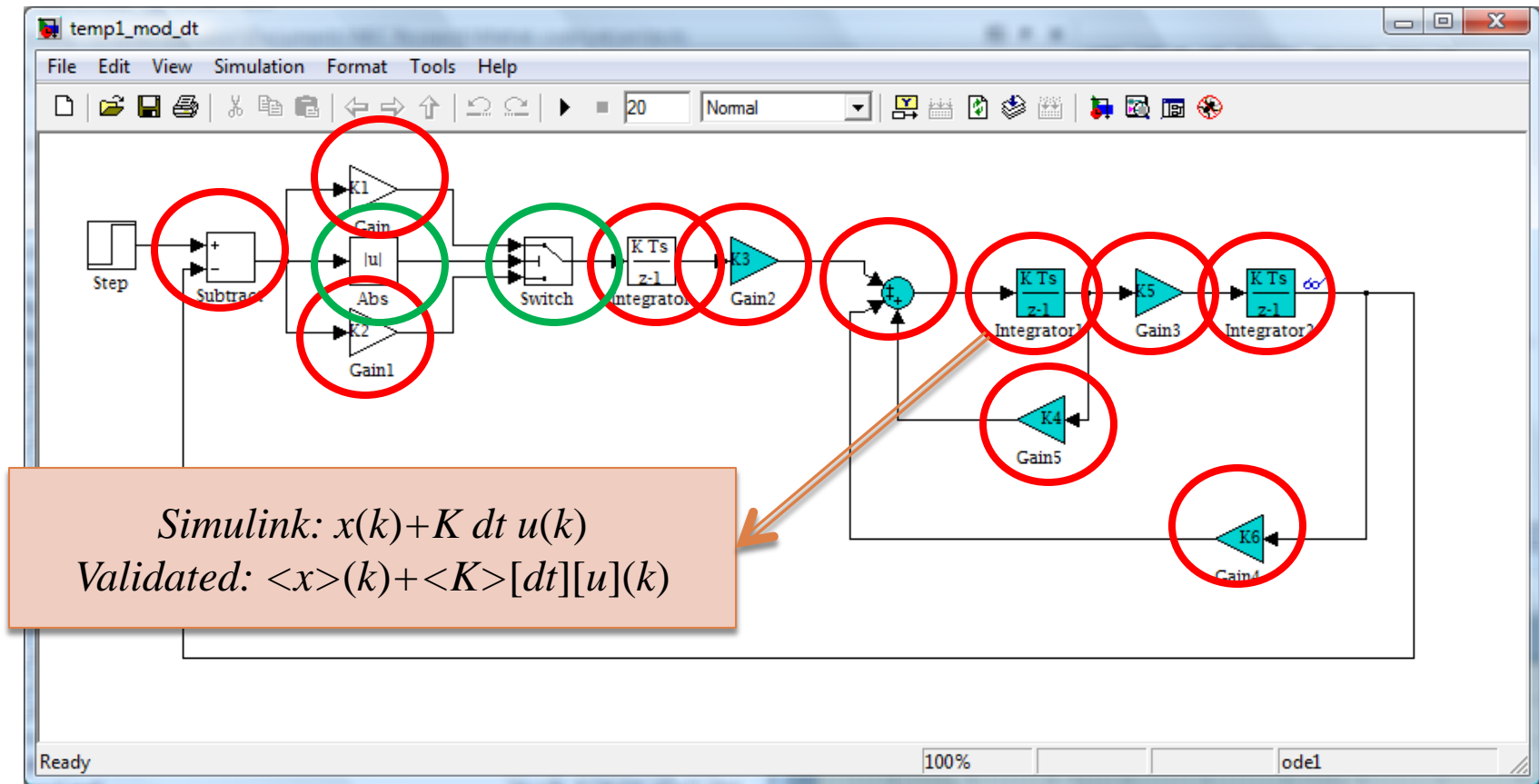
How to do validated computations on discrete-time models.

28



How to do validated computations on discrete-time models.

29

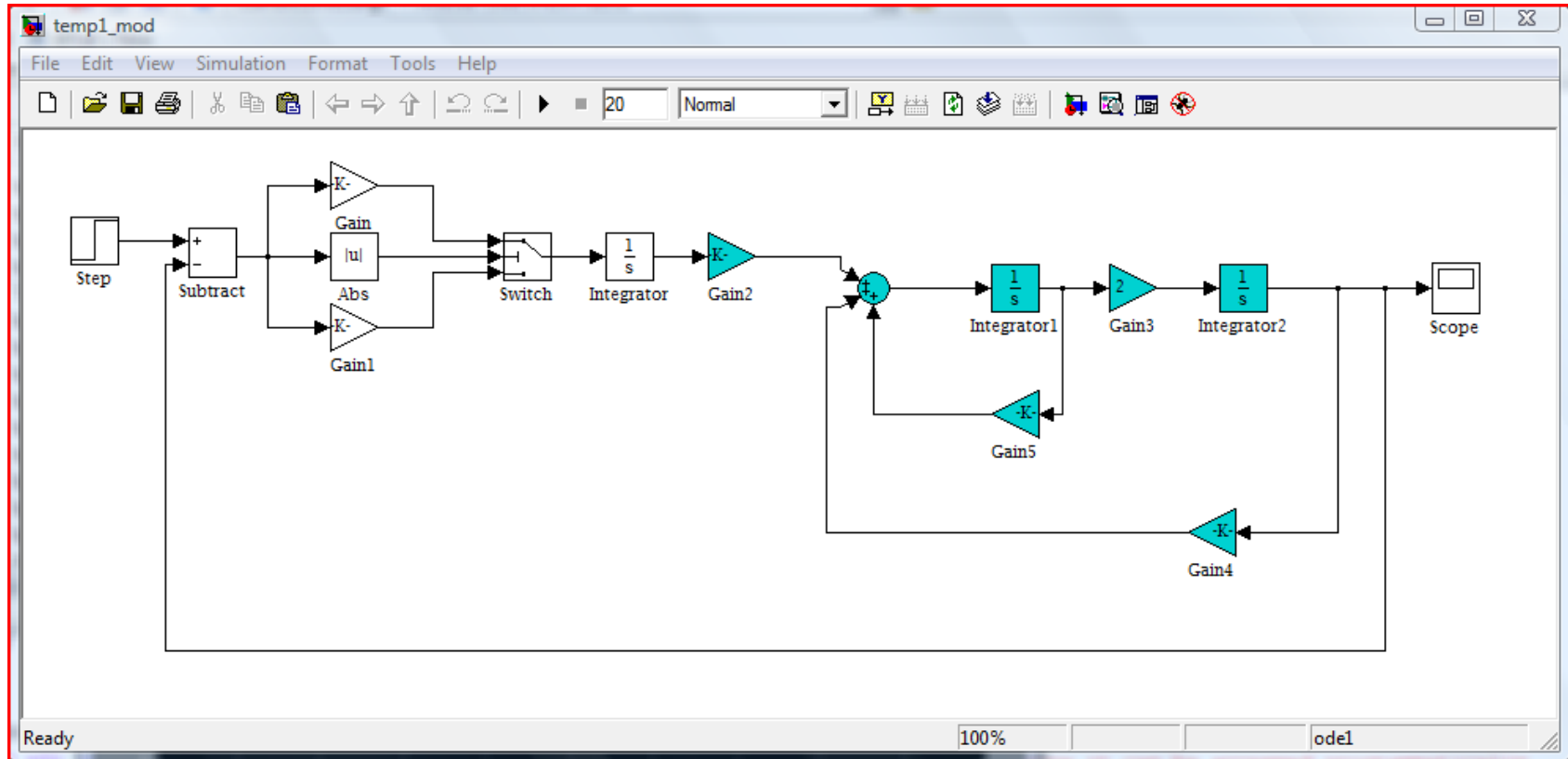


Red: we repeat the Simulink operation using validated arithmetics

Green: we compare numerical values and validated quantities and we record any potential violations

How to do validated computations on continuous-time models.

30



Same process as with discrete-time models,
but now we do symbolic computations.
Currently, the model must be linear.

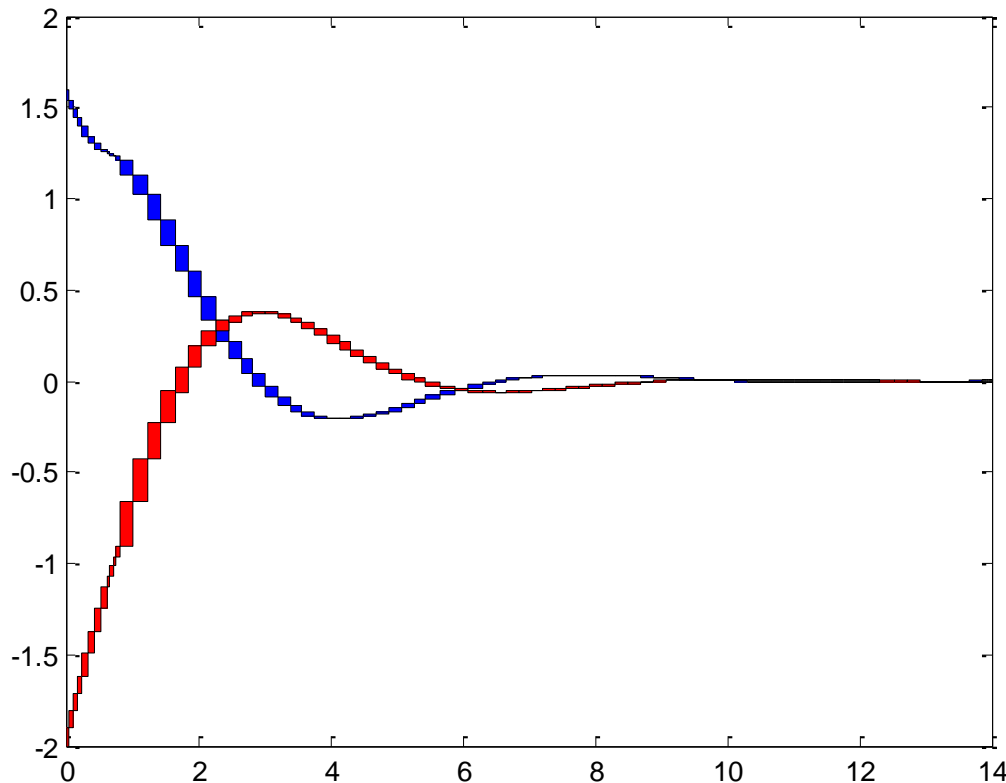
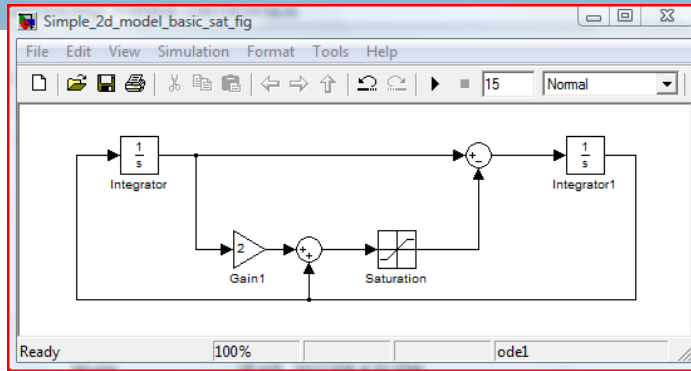
How to do validated computations on continuous-time models.

31

- From the symbolic computation, we derive
 - the current system dynamics
 - $dx/dt = A(p)x + B(p)u + v(p)$
 - the current state constraints:
 - $g(x, p, u) \leq 0$
- The state of the system at the next time step will be
 - $\langle x' \rangle = e^{[A][dt]} \langle x \rangle$ (ignore inputs)
 - For the inputs see Section IV.B in the paper
- Similarly, we can compute an enclosure for the system states between two time samples
 - The enclosure is used for checking the current state constraints

Continuous-time Example

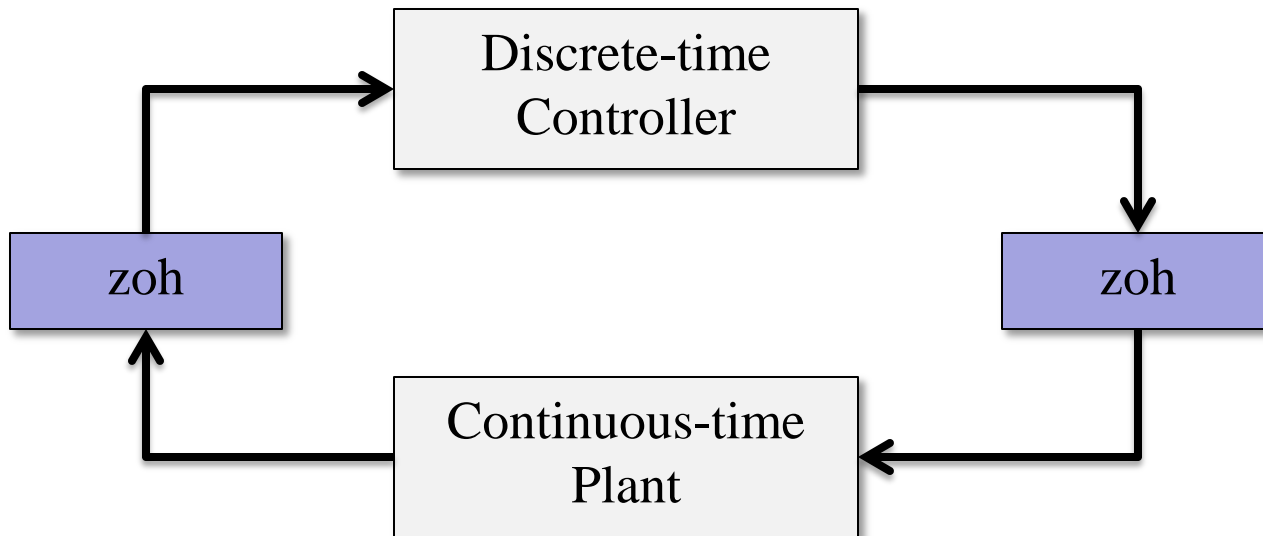
32



- Solution using ODE45 solver
- Variable step-size
- Total of 84 sim. points
- 14 warnings
 - 2 comparison warnings
 - 12 integration errors
- Example:
 - $t \approx 2.26$ or $i=20$
 - Using verified integration we detect that there might be saturation:
 $1 \in [0.7132, 1.0130]$
 - But in numerical simulation no saturation occurs:
 $i=19: 0.8453 < 1$
 $i=20: 0.8801 < 1$

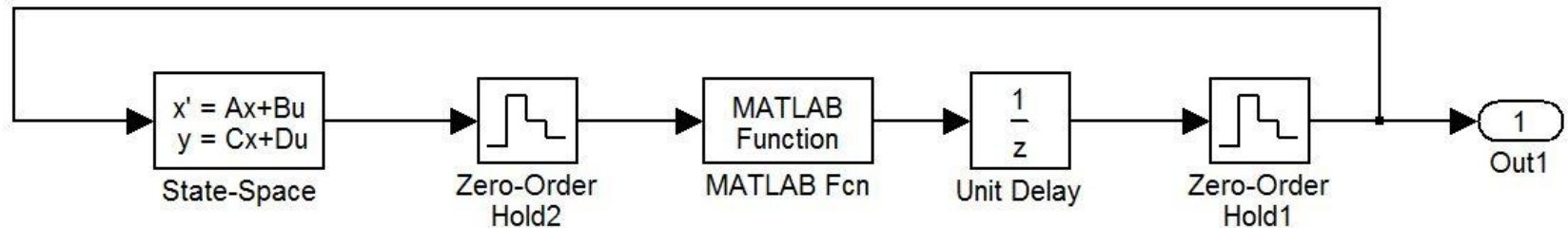
Mixed-Signal Systems

33



Mixed-Signal Example

34

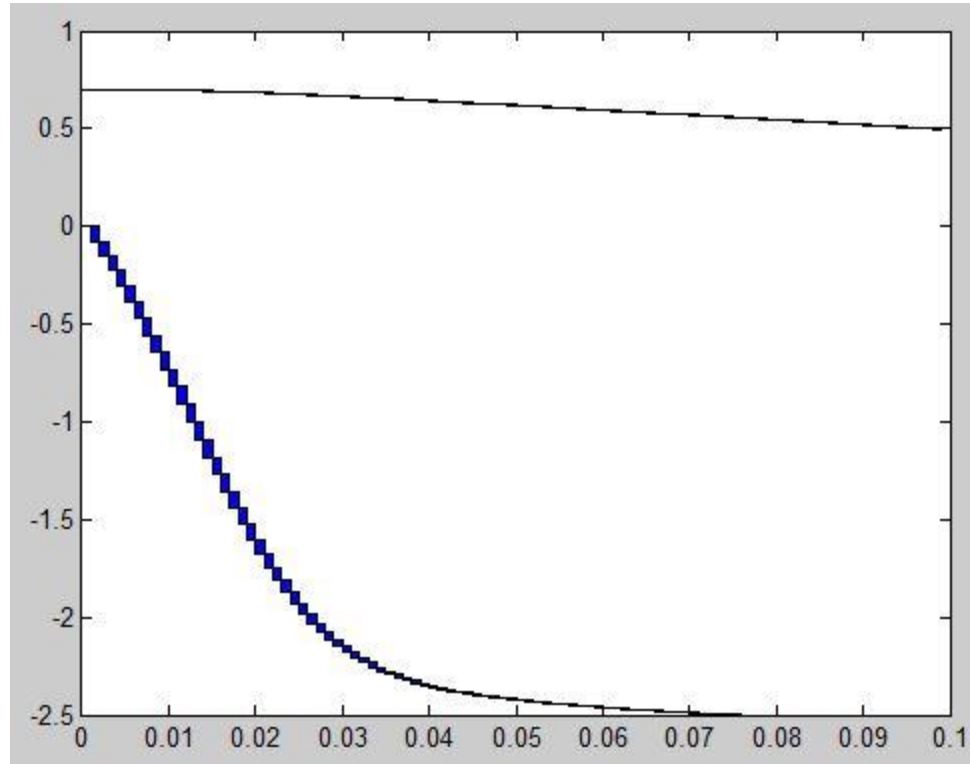


$$\dot{x}(t) = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} x(t) + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u(t)$$

$$\begin{aligned} F(x) = & -297.63x_1 - 31.92x_2 + 203.56x_1^2 + 87.34x_1x_2 - 6.58x_2^2 + 132.35x_1^3 \\ & + 142.99x_1^2x_2 + 12.74x_1x_2^2 - 9.22x_2^3 - 48.34x_1^4 - 199.35x_1^3x_2 \\ & + 7.92x_1^2x_2^2 - 3.74x_1x_2^3 - 1.17x_2^4. \end{aligned}$$

Mixed-Signal Example

35



RobSim: Developed Components

36

- Simulink support for validated computations
 - ▣ Most of the basic blocks are currently supported
 - ▣ Implementation by overloading operators
 - Any arithmetic implemented in Matlab can be used (intval, affnum, intnum).
 - Allows seamless integration of other arithmetics in the future, for example, probabilistic arithmetics
- Affine Arithmetic Toolbox (AffNum) for Matlab
 - ▣ Built upon IntLab - a toolbox for Interval Computations by Prof. Siegfried M. Rump (Hamburg University of Technology & Waseda University)
 - ▣ Full support of affine operations
 - ▣ Most commonly used non-linear functions in Matlab are supported
- Validated Solution of Linear Ordinary Differential Equations
 - ▣ Takes into account floating point rounding errors, uncertain parameters and sets of initial conditions

RobSim: Improvements and Additions





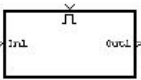


37

- Heuristics for interchange of computations in IA and AA
 - ▣ Improve speed while achieving good accuracy
- Validated integration for nonlinear systems
 - ▣ Current support only for linear systems
 - ▣ This requires the derivation of the mathematical model of the system through symbolic manipulations
- Validated computation of the exponential of a matrix
 - ▣ Currently is done through Taylor expansion
- Support for more Simulink blocks and Stateflow
 - ▣ Required for industrial size examples
 - ▣ the Simulink block library is large!

Appendix




Common blocks supported


39

Blocks	
	Abs: It outputs the absolute value of the input signal. There are no parameters to set.
	Gain: This is multiplication with a constant. The constant can be a semi-symbolic value defined through the parameter map.
	Product: This blocks multiplies the input signals. Any number of input signals under multiplication are supported. The product block should be used with <i>caution</i> ; since if it multiplies two continuous states of a model, then it will create a nonlinear system.
	Saturation: It saturates the input signal. The upper and lower limits can only be numeric values.
	Subsystem: It is used to group Simulink blocks. Only two subsystems are currently supported: subsystems with only input and output ports and subsystems with an enable port.
	Sum: This block sums the values of the input signals. Any number of input signals under addition or subtraction are supported.
	Switch: It switches between two input signals according to the value of a third signal. The switching threshold can only be a numeric value.

Common blocks supported

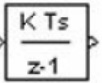
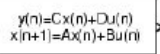
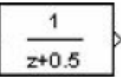

40

Sources	
	Constant: The constant value that the block outputs can be a semi-symbolic value defined through the parameter map.
	Step: The block generates a step signal. The initial and final values that the block outputs can be semi-symbolic values defined through the parameter map. The step time cannot be a range.
	Uniform Random Number: In Simulink models, this block, when sampled, picks a random number from a uniform distribution. In ROBSIM, the output of the block is an interval with its minimum and maximum values defined by the respective parameters of the block.

Sinks	
	Output: There are no parameters. The values of the port at each time step are stored and outputted in one of the outputs variables of ROBSIM.


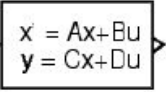
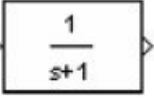
Discrete-time blocks

41

States	
	Discrete integrator: The block performs an integration step. The block contains a gain which can be a semi-symbolic value defined through the parameter map. The time step for the integration is provided by the simulation.
	Discrete state space: This block implements a discrete-time linear system given its state space representation. The system matrices can contain semi-symbolic values through parameters defined in the parameter map.
	Discrete transfer function: This block implements a discrete-time transfer function. The numerator and denominator can contain semi-symbolic values through parameters defined in the parameter map.
	Unit delay: The block implements a delay for one time step. There are no parameters to set.

Continuous-time blocks

42

States	
	Integrator: The block performs integration of a continuous-time signal. There are no parameters to set.
	State space: This block implements a linear system given its state space representation. The system matrices can contain semi-symbolic values through parameters defined in the parameter map.
	Transfer function: This block implements a transfer function. The numerator and denominator can contain semi-symbolic values through parameters defined in the parameter map.