Towards an Operational Semantics of the Simulation Engine of Simulink

Alexandre Chapoutot

joint work with Olivier Bouissou (CEA LIST, LMeASI)

ENSTA Paristech

SYNCHRON 2011
December 1st

Motivation

Simulink and its industrial uses

Context

Simulink is a `de facto` standard in the industry to design embedded safety critical applications.
e.g. it is used to design almost all the **drive-by-wire** systems.

Components of a drive-by-wire systems

- A **physical mechanism** we want to control, e.g. a suspension.
- A **software** which implement the controller.

$\Rightarrow$ we have to deal with **hybrid systems**.

Current validation method

The main method to validate the design of embedded systems is based on **simulation activity**.

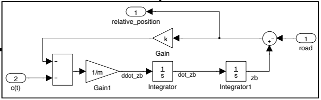
Classical design and validation methodology

Identification

Mathematical model

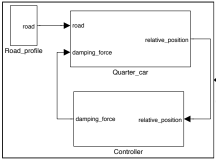
$$m\ddot{x} + c\dot{x} + kx = u$$

Computer description



Theoretical and experimental studies

Controller definition



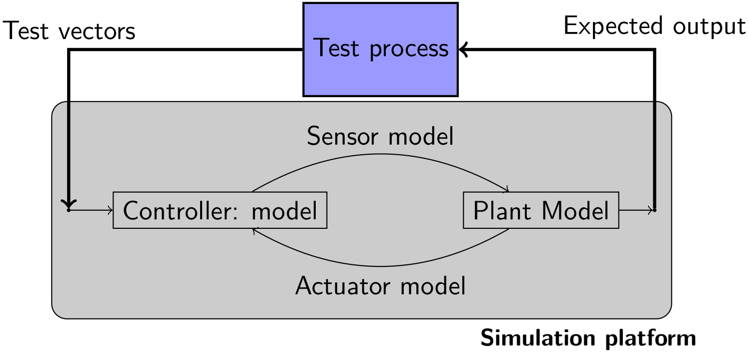
Remarks

- **Mathematical models** are an approximated descriptions of **physical systems**.
- The **computer descriptions** are studied with numerical tools.

Classical design and validation methodology

MIL: Model in the loop

Definition of a mathematical model of the `plant` and the `controller`.



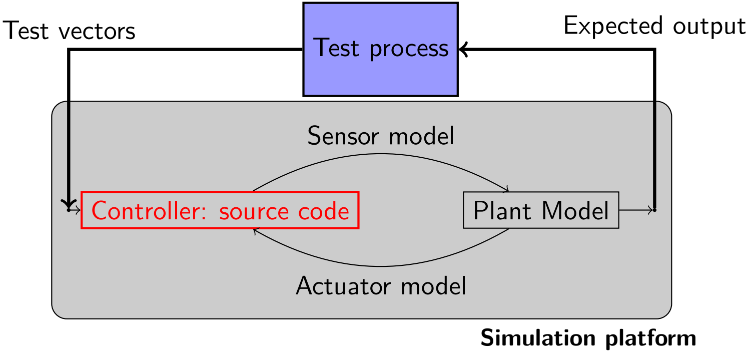
Aim of the testing activity

- Does the controller fulfil the specification?
- The set of tests will be used as "an oracle" in the next steps.

Classical design and validation methodology

SIL: Software in the loop

Implementation of the `controller` in a target language.



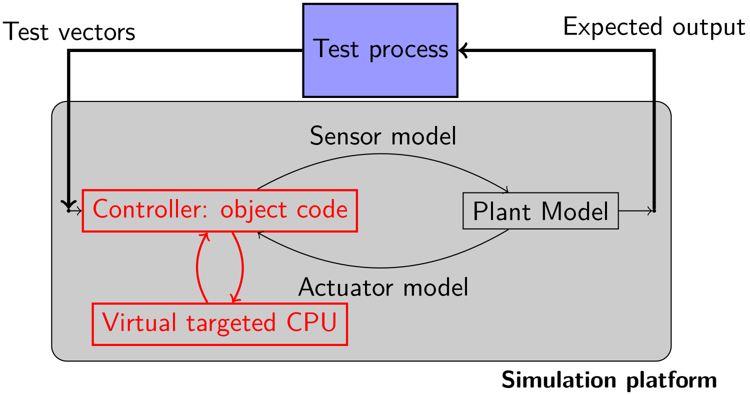
Aim of the testing activity

- Do the hand-written or generated code still fulfil the specification?

Classical design and validation methodology

PIL: Processor in the loop

Compilation of the controller and execute it on a virtual processor.



Aim of the testing activity

- Does the low-level implementation still fulfil the specification?

Considering Simulink as a language

Simulink: a graphical language

It allows to describe, as block-diagrams, a mixing of:

- **Ordinary differential equations** (ODE).
- Finite difference equations: **single-rate** or **multi-rate** period.
- Conditional executed equations: **enabled** or **triggered**.

Simulink: several parametrized semantics

- the definition of a **numerical solver** used to solve ODEs;
- the detection of events: **zero-crossing**.

So, it is a **numerical approximation** of the **mathematical behavior**.

What we should keep in mind!

The **designer** only refers to the **Simulink's semantics** (i.e. graphical output) to **validate** the development steps!

Formal verification of Simulink

Fact: simulations will never be complete and cannot bring strong guarantee on the designed software.

Main question: How can we help designer to be more confident in its designed systems with formal methods?

Formal verification of Simulink: **Our Goal**

- To compute invariant properties on the `software` **taking into account** `a model of the plant`.
- **Our approach:** abstract interpretation-based static analysis.

Formal verification of Simulink: **Constraint**

- Formal verification methods should be adaptable for the design process in order to be more easily adopted by the end-user.

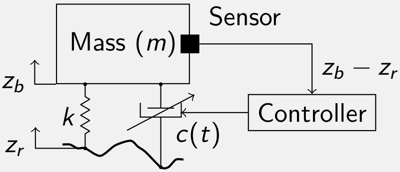
Formal verification of Simulink: **Challenge**

- Understand the **simulation engine** of Simulink.

Simulink Language

A simple example: mathematical models

A semi-active suspension of a quarter-car model



- $m = 250$ kg
- $k = 20000$ N/m
- $c_{\max} = 16000$ N/m/s
- $c_{\min} = 0$

Mathematical model of the mechanical system

$$\ddot{z}_b = -\frac{1}{m}\Big(k(z_b - z_r) + c(t)\Big) \ .$$

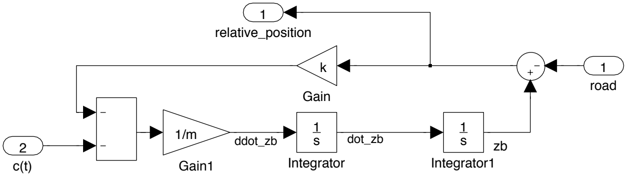
Mathematical model of the controller

$$c(t) = \begin{cases} -c_{\max}(z_b - z_r) & \text{if } (z_b - z_r)(\dot{z}_b - \dot{z}_r) < 0 \\ c_{\min} & \text{if } (z_b - z_r)(\dot{z}_b - \dot{z}_r) \geq 0 \end{cases} \ .$$

A simple example: Simulink implementation – 1

Mathematical model of the mechanical system

$$\ddot{z}_b = -\frac{1}{m}\Big(k(z_b - z_r) + c(t)\Big) \quad .$$



Integrator block

Associated to a first order dynamic system:

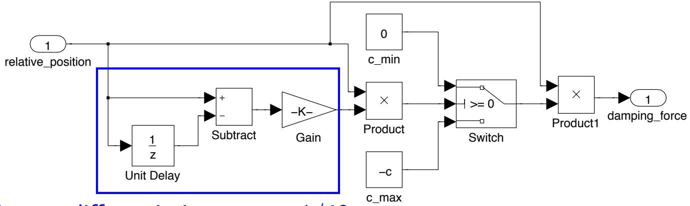
$$\begin{cases} \dot{x}(t) = \mathrm{input}(t) \\ \mathrm{output}(t) = x(t) \end{cases} \quad \text{with} \quad x(0) = x_0$$

A simple example: Simulink implementation – 2

Mathematical model of the controller

$$c(t) = \begin{cases} -c_{\max}(z_b - z_r) & \text{if } (z_b - z_r)(\dot{z}_b - \dot{z}_r) < 0 \\ c_{\min} & \text{if } (z_b - z_r)(\dot{z}_b - \dot{z}_r) \geq 0 \end{cases}.$$

Implementation: $(\dot{z}_b - \dot{z}_r)$ is given by differentiating the sensor output.



Discrete differentiation at rate $1/40$ sec.

Closed-loop system

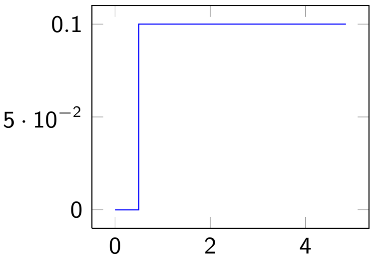
- Connect the output of the plant to the input of the controller.
- Connect the output of the controller to the input of the plant.

A simple example: simulation results

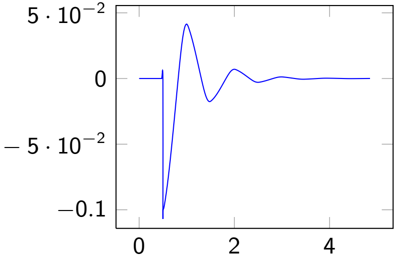
Simulation settings

- Duration of the simulation: 5 seconds.
- Variable step-size solver: `ode23`.
- Zero-crossing detection activated (non adaptive version).

Road profile



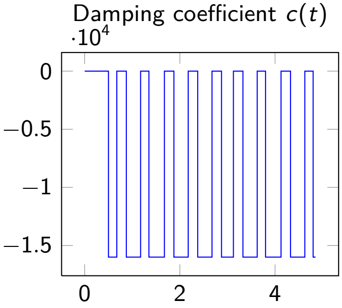
Relative position

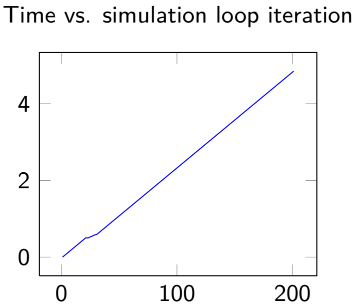


Remarks

- The on-off controller make the suspension stable.

A simple example: simulation results





Remark: numerical precision

The damping force oscillated because of tiny variations of the suspension.
Consequence: finite precision may induce unexpected behaviors.

Remark: time

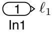
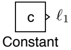
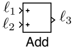
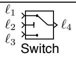
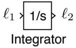
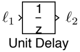
The time evolution is not homogeneous:
Consequence: blocks depending on time may produce more or less output following the time evolution e.g. sine block.

Operational Semantics

Simulink as an equation-based programming language

Input/Output relation

Each block defines the time invariant relation between its input and its output.

Library	Blocks	Representation	Equations
Sources	Input		$\ell_1 = \text{in1}$
	Constant		$\ell_1 = c$
Arithmetic	Add		$\ell_3 = \ell_1 + \ell_2$
Signal routing	Switch		$\ell_4 = \text{if}\,(p_r(\ell_2), \ell_1, \ell_3)$
Continuous-time	Integrator		$\{\ell_2 = x; \dot{x} = \ell_1; x(0) = \text{init}\}$
Discrete-time	Unit Delay		$\{\ell_2 = d; \bar{d} =_S \ell_1; d(0) = \text{init}\}$

$\bar{d} =_S \ell$ stands for at each $t = kS$, $d = \ell$ else it keeps its previous value.

Simulink as an equation-based programming language

Input/Output relation

Each block defines the time invariant relation between its input and its output.

A core language of equations

$$e ::= r \mid \ell \mid x \mid d \mid e_1 \diamond e_2 \mid e_1 \bowtie e_2 \mid \mathrm{if}\big(e_1, e_2, e_3\big) \quad (1)$$

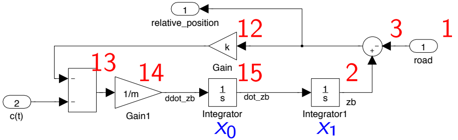
$$eq ::= \ell :=_S e \mid \ell := e \mid \dot{x} := e \mid \bar{d} :=_S e \quad (2)$$

$$p ::= eq \mid eq; p \quad (3)$$

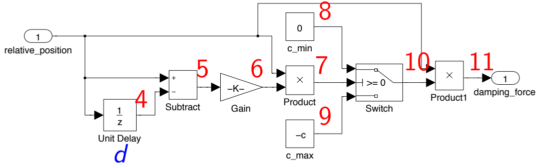
with

- $r \in \mathbb{R}$, constant values;
- $\ell \in \mathcal{V}$, variables associated to a **block output**;
- $x \in \mathcal{V}$, variables associated to **continuous-time states**;
- $d \in \mathcal{V}$, variables associated to **discrete-time states**;
- $\diamond \in \{+, -, \times, \div\}$, arithmetic operations;
- $\bowtie \in \{<, \leq, >, \geq, =, <>\}$, relational operations;
- S is the set of all the sampling times.

Simulink as an equation-based programming language



In red: evaluation order of blocks



For each block in the evaluation order, a simple translation gives:

$$
\begin{aligned}
\ell_1 &= \text{input} \\
\dot{x}_1 &= \ell_{15} \\
\ell_2 &= x_1 \\
\ell_3 &= \ell_2 - \ell_1 \\
\bar{d} &=_S \ell_3 \\
\ell_4 &= d
\end{aligned}
$$

$$
\begin{aligned}
\ell_5 &= \ell_3 - \ell_4 \\
\ell_6 &= 1/40 \times \ell_5 \\
\ell_7 &= \ell_3 \times \ell_6 \\
\ell_8 &= 0 \\
\ell_9 &= -16000 \\
\ell_{10} &= \text{if } \ell_7 \geq 0 \text{ then } \ell_8 \text{ else } \ell_9
\end{aligned}
$$

$$
\begin{aligned}
\ell_{11} &= \ell_3 \times \ell_{10} \\
\ell_{12} &= 20000 \times \ell_3 \\
\ell_{13} &= -\ell_{12} - \ell_{11} \\
\ell_{14} &= 1/250 \times \ell_{13} \\
\dot{x}_0 &= \ell_{14} \\
\ell_{15} &= x_0
\end{aligned}
$$

Simulink as an equation-based programming language

Kinds of equations

A Simulink model is made of four kinds of functions:

- the output function;
- the update function of discrete-time states;
- the update function of continuous-time states;

$$
\begin{aligned}
\ell_1 &= \text{input} \\
\dot{x}_1 &= \ell_{15} \\
\ell_2 &= x_1 \\
\ell_3 &= \ell_2 - \ell_1 \\
\bar{d} &=_S \ell_3 \\
\ell_4 &= d
\end{aligned}
\qquad
\begin{aligned}
\ell_5 &= \ell_3 - \ell_4 \\
\ell_6 &= 1/40 \times \ell_5 \\
\ell_7 &= \ell_3 \times \ell_6 \\
\ell_8 &= 0 \\
\ell_9 &= -16000 \\
\ell_{10} &= \text{if } \ell_7 \geq 0 \text{ then } \ell_8 \text{ else } \ell_9
\end{aligned}
\qquad
\begin{aligned}
\ell_{11} &= \ell_3 \times \ell_{10} \\
\ell_{12} &= 20000 \times \ell_3 \\
\ell_{13} &= -\ell_{12} - \ell_{11} \\
\ell_{14} &= \frac{1}{250} \times \ell_{13} \\
\dot{x}_0 &= \ell_{14} \\
\ell_{15} &= x_0
\end{aligned}
$$

Remark

These functions allow a **state-space representation** of a model.

Simulink as an equation-based programming language

Kinds of equations

A Simulink model is made of four kinds of functions:

- the output function;
- the update function of discrete-time states;
- the update function of continuous-time states;

$\ell_1 = \text{input}$

$\dot{x}_1 = \ell_{15}$

$\ell_2 = x_1$

$\ell_3 = \ell_2 - \ell_1$

$\bar{d} =_S \ell_3$

$\ell_4 = d$

$\ell_5 = \ell_3 - \ell_4$

$\ell_6 = 1/40 \times \ell_5$

$\ell_7 = \ell_3 \times \ell_6$

$\ell_8 = 0$

$\ell_9 = -16000$

$\ell_{10} = \text{if } \ell_7 \geq 0 \text{ then } \ell_8 \text{ else } \ell_9$

$\ell_{11} = \ell_3 \times \ell_{10}$

$\ell_{12} = 20000 \times \ell_3$

$\ell_{13} = -\ell_{12} - \ell_{11}$

$\ell_{14} = \frac{1}{250} \times \ell_{13}$

$\dot{x}_0 = \ell_{14}$

$\ell_{15} = x_0$

Notation

We denote this function by $\mathbf{g}(\mathbf{t}, \mathbf{x}, \mathbf{d})$.

Simulink as an equation-based programming language

Kinds of equations

A Simulink model is made of four kinds of functions:

- the output function;
- the update function of discrete-time states;
- the update function of continuous-time states;

$\ell_1 = \text{input}$

$\dot{x}_1 = \ell_{15}$

$\ell_2 = x_1$

$\ell_3 = \ell_2 - \ell_1$

$\bar{d} =_S \ell_3$

$\ell_4 = d$

$\ell_5 = \ell_3 - \ell_4$

$\ell_6 = 1/40 \times \ell_5$

$\ell_7 = \ell_3 \times \ell_6$

$\ell_8 = 0$

$\ell_9 = -16000$

$\ell_{10} = \text{if } \ell_7 \geq 0 \text{ then } \ell_8 \text{ else } \ell_9$

$\ell_{11} = \ell_3 \times \ell_{10}$

$\ell_{12} = 20000 \times \ell_3$

$\ell_{13} = -\ell_{12} - \ell_{11}$

$$\ell_{14} = \frac{1}{250} \times \ell_{13}$$

$\dot{x}_0 = \ell_{14}$

$\ell_{15} = x_0$

Notation

We denote this function by $\mathbf{f_d}(\mathbf{t}, \mathbf{x}, \mathbf{d}) = g(t, x, d) \mid_{\bar{d} =_S \ell}$.

Simulink as an equation-based programming language

Kinds of equations

A Simulink model is made of four kinds of functions:

- the output function; (2 versions: major g and **minor** $\tilde{g}$)
- the update function of discrete-time states;
- the update function of continuous-time states;

$$\ell_1 = \text{input}$$
$$\dot{x}_1 = \ell_{15}$$
$$\ell_2 = x_1$$
$$\ell_3 = \ell_2 - \ell_1$$
$$\bar{d} =_S \ell_3$$
$$\ell_4 = d$$

$$\ell_5 = \ell_3 - \ell_4$$
$$\ell_6 = 1/40 \times \ell_5$$
$$\ell_7 = \ell_3 \times \ell_6$$
$$\ell_8 = 0$$
$$\ell_9 = -16000$$
$$\ell_{10} = \text{if } \ell_7 \geq 0 \text{ then } \ell_8 \text{ else } \ell_9$$

$$\ell_{11} = \ell_3 \times \ell_{10}$$
$$\ell_{12} = 20000 \times \ell_3$$
$$\ell_{13} = -\ell_{12} - \ell_{11}$$
$$\ell_{14} = \frac{1}{250} \times \ell_{13}$$
$$\dot{x}_0 = \ell_{14}$$
$$\ell_{15} = x_0$$

Notation

We denote this function by $\mathbf{f_x}(\mathbf{t}, \mathbf{x}, \mathbf{d}) = \tilde{g}(t, x, d) \mid_{\dot{d} =_S \ell}$.

Simulink as an equation-based programming language

Kinds of equations

A Simulink model is made of four kinds of functions:

- the output function;
- the update function of discrete-time states;
- the update function of continuous-time states;

Hidden equations: the 5th kind

Some blocks are associated to equations to detect **zero-crossing events** (only with variable step solvers).

Example

`Switch block`: detect when the sign of the 2nd input changes.

Notation

We denote a zero-crossing equation $f_z(t, x, d)$.

Overview of numerical simulation

Goal: computing the **temporal evolution** of the system.

The steps of the Simulink's simulation engine

Input: $\mathbf{x}_0$, $\mathbf{d}_0$, t_0, h_0;
$n = 0$;
loop until $t_n \geq t_{\text{end}}$
    evaluate $g(t_n, \mathbf{x}_n, \mathbf{d}_n)$;
    update $\mathbf{d}' = f_d(t_n, \mathbf{x}_n, \mathbf{d}_n)$;
    solve $\dot{\mathbf{x}}(t) = f_x(t, \mathbf{x}(t), \mathbf{d}_n)$ over interval $[t_n, t_n + h_n]$ to get $\mathbf{x}(t_n + h_n)$;
    find_zero_crossing ;
    compute h_{n+1} ;
    compute t_{n+1} ;
    $\mathbf{d}_{n+1} = \mathbf{d}'$; $\mathbf{x}_{n+1} = \mathbf{x}(t_n + h_n)$; $n = n + 1$;
end loop

Remarks

- Major steps: evaluation of g produces the simulation results at t_n.
- Minor steps: evaluations of $\tilde{g}$ are used as intermediate computations between t_n and $t_n + h_n$.

Overview of numerical simulation

The **temporal evolution** of the system depends on a set of parameters.

Parameters (a sample)

A Simulink model is described by a set of parameters:

- $t_0 = 0$ **start time** of the simulation;
- $t_{\text{end}} = 10$ **stop time** of the simulation;
- **numerical integration methods** with an **absolute tolerance** ($\texttt{atol} = 10^{-6}$), a **relative tolerance** ($\texttt{rtol} = 10^{-3}$);
- **minimal** ($\texttt{hmin} = 0.2$) and **maximal** ($\texttt{hmax} = 5$) integration step-size;
- **zero-crossing method**: adaptive or non adaptive;
- zero-crossing tolerance ($\texttt{zctol} = 10 \times 128 \times \text{eps}$).

Notation

All these parameters are represented by a record π : name $\rightarrow$ value.

Consequence

There are several semantics of Simulink.

Operational semantics of Simulink

...

loop until $t_n \geq t_{\text{end}}$

  evaluate $g(t_n, \mathbf{x}_n, \mathbf{d}_n)$;

  update $\mathbf{d}' = f_d(t_n, \mathbf{x}_n, \mathbf{d}_n)$;

  solve $\dot{\mathbf{x}}(t) = f_x(t, \mathbf{x}(t), \mathbf{d}_n)$ over interval $[t_n, t_n + h_n]$ to get $\mathbf{x}(t_n + h_n)$;

  find_zero_crossing ;

  ...

end loop

Main rules

$$\frac{\sigma(t) = \pi(t_{\text{end}})}{\mathrm{Eq}, \pi \vdash \sigma \Rightarrow \sigma} \ \textsc{Simulation-End}$$

$$\frac{\sigma(t) < \pi(t_{\text{end}}) \qquad \mathrm{Eq}, \pi \vdash \sigma \xrightarrow{M} \sigma_1 \qquad \mathrm{Eq}, \pi \vdash \sigma_1 \xrightarrow{u} \sigma_2 \qquad \mathrm{Eq}, \pi \vdash \sigma_2 \xrightarrow{s} \sigma'}{\mathrm{Eq}, \pi \vdash \sigma \Rightarrow \sigma'}$$

Notation

$$\mathrm{Eq} = \left\{ g(t, x, d), f_d(t, x, d), f_x(t, x, d), f_z(t, x, d) \right\}$$

Rules for output function

Recall: main rules

$$\frac{\sigma(t) < \pi(t_{\mathrm{end}}) \qquad \mathrm{Eq}, \pi \vdash \sigma \xrightarrow{\mathsf{M}} \sigma_1 \qquad \mathrm{Eq}, \pi \vdash \sigma_1 \xrightarrow{u} \sigma_2 \qquad \mathrm{Eq}, \pi \vdash \sigma_2 \xrightarrow{s} \sigma'}{\mathrm{Eq}, \pi \vdash \sigma \Rightarrow \sigma'}$$

Very simple rules of a classical operational semantics.

Step

- Given values of continuous states x , values of discrete states d and input values.
- Evaluate each equations given by $g(t, x, d)$ in the evaluation order

Sample of evaluation rules

$$\frac{\langle e_1, \sigma \rangle \xrightarrow{o} r_1 \qquad \langle e_2, \sigma \rangle \xrightarrow{o} r_2}{\langle e_1 \diamond e_2, \sigma \rangle \xrightarrow{o} r_1 \diamond r_2} \text{ ARITH} \qquad \frac{\langle e_1, \sigma \rangle \xrightarrow{o} \mathsf{true} \qquad \langle e_2, \sigma \rangle \xrightarrow{o} r_2}{\langle \mathsf{if}\ (e_1, e_2, e_3), \sigma \rangle \xrightarrow{o} r_2} \text{ THEN}$$

Rules for update discrete states

Recall: main rules

$$\frac{\sigma(t) < \pi(t_{\text{end}}) \qquad \text{Eq}, \pi \vdash \sigma \xrightarrow{M} \sigma_1 \qquad \text{Eq}, \pi \vdash \sigma_1 \xrightarrow{\mathbf{u}} \sigma_2 \qquad \text{Eq}, \pi \vdash \sigma_2 \xrightarrow{s} \sigma'}{\text{Eq}, \pi \vdash \sigma \Rightarrow \sigma'}$$

Main idea:

- if the current time **t** is a sampling time then update the state.
- otherwise keep the previous value.

Simple rules

$$\frac{\text{Eq}(\mathbf{d}) = \overline{\mathbf{d}} :=_S \ell \qquad \sigma(t) \in S}{\text{Eq}, \pi \vdash \sigma \xrightarrow{u} \sigma[\mathbf{d} \mapsto \sigma(\ell)]} \qquad \frac{\text{Eq}(\mathbf{d}) = \overline{\mathbf{d}} :=_S \ell \qquad \sigma(t) \notin S}{\text{Eq}, \pi \vdash \sigma \xrightarrow{u} \sigma}$$

Remark

The numerical integration is in charge not to miss a sampling time.

Rules for update continuous states

Recall: Main rules

$$\frac{\sigma(t) < \pi(t_{\text{end}}) \qquad \text{Eq}, \pi \vdash \sigma \xrightarrow{M} \sigma_1 \qquad \text{Eq}, \pi \vdash \sigma_1 \xrightarrow{u} \sigma_2 \qquad \text{Eq}, \pi \vdash \sigma_2 \xrightarrow{s} \sigma'}{\text{Eq}, \pi \vdash \sigma \Rightarrow \sigma'}$$

The main features of the Simulink solver is encoded in this rules:

- Performing numerical integration.
- Handling zero-crossing.
- Adapting integration step-size.

Sub rules

$$\frac{\text{Eq}, \pi, \sigma \vdash \sigma \xrightarrow{i} \sigma_{so} \qquad \text{Eq}, \pi, \sigma \vdash \sigma_{so} \xrightarrow{zc} \sigma_{zc} \qquad \text{Eq}, \pi \vdash \sigma_{zc} \xrightarrow{h} \sigma'}{\text{Eq}, \pi \vdash \sigma \xrightarrow{s} \sigma'}$$

Remarks

Hypotheses to enforce the existence of the ODEs solution.

- No discrete states are updated during numerical integration.
- Numerical integration assumes no zero-crossing events appear.

Numerical integration methods – 1

There exists a huge amount of methods which may categorize in:

- explicit or implicit method;
- fixed or variable step-size;
- single-step or multi-step method;
- fixed or variable order.

Simulink's integration methods considered

Explicit single-step fixed step-size

- ode1 (Euler)
- ode2 (Heun)
- ode3 (Bogacki-Shampine)
- ode4 (Dormand-Prince)
- ode8 (Dormand-Prince)

Explicit single-step variable step-size

- ode23 (Bogacki-Shampine)
- ode45 (Dormand-Prince)

Remark

All these methods are members of the Runge-Kutta family which has a unified representation with **Butcher Table**.

Numerical integration methods – 2

Example

Simulink solver `ode23`

Mathematical description

$$k_1 = f(t_n, x_n)$$

$$k_2 = f(t_n + \tfrac{1}{2} h_n, x_n + \tfrac{1}{2} h k_1)$$

$$k_3 = f(t_n + \tfrac{3}{4} h_n, x_n + \tfrac{3}{4} h k_2)$$

$$x_{n+1} = x_n + h\left(\tfrac{2}{9} k_1 + \tfrac{1}{3} k_2 + \tfrac{4}{9} k_3\right)$$

$$k_4 = f(t_n + 1\, h_n, x_{n+1})$$

$$y_{n+1} = x_n + h\left(\tfrac{7}{24} k_1 + \tfrac{1}{4} k_2 + \tfrac{1}{3} k_3 + \tfrac{1}{8} k_4\right)$$

Butcher Table

$$\begin{array}{c|cccc}
0 & & & & \\
\frac{1}{2} & \frac{1}{2} & & & \\
\frac{3}{4} & 0 & \frac{3}{4} & & \\
1 & \frac{2}{9} & \frac{1}{3} & \frac{4}{9} & \\
\hline
 & \frac{2}{9} & \frac{1}{3} & \frac{4}{9} & 0 \\
 & \frac{7}{24} & \frac{1}{4} & \frac{1}{3} & \frac{1}{8}
\end{array}$$

Remark

The parameters set π embeds the Butcher Table of the considered integration method $\Rightarrow$ **handling of the several semantics of Simulink**.

Numerical integration methods – 2

Example

Simulink solver `ode23`

Mathematical description

$$k_1 = f(t_n, x_n)$$
$$k_2 = f(t_n + \tfrac{1}{2} h_n, x_n + \tfrac{1}{2} h k_1)$$
$$k_3 = f(t_n + \tfrac{3}{4} h_n, x_n + \tfrac{3}{4} h k_2)$$
$$x_{n+1} = x_n + h\left(\tfrac{2}{9} k_1 + \tfrac{1}{3} k_2 + \tfrac{4}{9} k_3\right)$$
$$k_4 = f(t_n + 1 h_n, x_{n+1})$$
$$y_{n+1} = x_n + h\left(\tfrac{7}{24} k_1 + \tfrac{1}{4} k_2 + \tfrac{1}{3} k_3 + \tfrac{1}{8} k_4\right)$$

Simulink translation

$$k_1 = f_x(t_n, x_n, d_n);$$
$$x = x_n + \tfrac{1}{2} k_1;\ t = t_n + \tfrac{1}{2} h_n;$$
$$k_2 = f_x(t, x, d_n);$$
$$x = x_n + \tfrac{3}{4} k_1;$$
$$\ldots$$
$$x_{n+1} = x_n + h\left(\tfrac{2}{9} k_1 + \tfrac{1}{3} k_2 + \tfrac{4}{9} k_3\right);$$
$$\ldots$$

Consequence

Our semantic rules act as an evaluation of a local expansion of the sequence of equations.

Numerical integration methods – 2

Evaluation rules for variable step-size solver

$$\frac{\begin{array}{c}\langle \mathtt{sc}_1(\sigma_M(\mathbf{x}), \mathsf{Eq}(\mathbf{x}), \pi)\ ,\ \sigma\rangle \xrightarrow{o} \sigma_1 \\ \langle \mathtt{sc}_2(\sigma_M(\mathbf{x}), \mathsf{Eq}(\mathbf{x}), \pi)\ ,\ \sigma_1\rangle \xrightarrow{o} \sigma_2 \qquad \mathtt{check\_err}(\sigma, \sigma_1, \sigma_2, \pi) = 1\end{array}}{\mathsf{Eq}, \pi, \sigma_M \vdash \sigma \xrightarrow{i} \sigma_M[\mathbf{x} \mapsto \sigma_1(\mathbf{x}),\ h \mapsto \sigma_1(h)]} \text{ INTEGR-SUCCESS}$$

$$\frac{\begin{array}{c}\langle \mathtt{sc}_1(\sigma_M(\mathbf{x}), \mathsf{Eq}(\mathbf{x}), \pi)\ ,\ \sigma\rangle \xrightarrow{o} \sigma_1 \\ \langle \mathtt{sc}_2(\sigma_M(\mathbf{x}), \mathsf{Eq}(\mathbf{x}), \pi)\ ,\ \sigma_1\rangle \xrightarrow{o} \sigma_2 \qquad \mathtt{check\_err}(\sigma, \sigma_1, \sigma_2, \pi) = 0 \\ \mathsf{Eq}, \pi, \sigma_M \vdash \sigma_M[h \mapsto \max(\pi(h_{\min}), \frac{\sigma(h)}{2})] \xrightarrow{i} \sigma'\end{array}}{\mathsf{Eq}, \pi, \sigma_M \vdash \sigma \xrightarrow{i} \sigma'} \text{ INTEGR-FAIL}$$

Note: sc_i represents the local expansion of equations.

Solving ODEs leads to a time discretization

Equations of the form $\dot{x}_i = \ell_j$ are substituted by the **sequence of the equations** needed to realize the numerical integration.

Numerical integration methods – 3

Validation of the integration step

For adaptive step-size method: for all continuous state variables

$$\text{integration error} = \frac{h_n \parallel x_{n+1} - y_{n+1} \parallel_\infty}{\max\left(\max\left(|x_{n+1}|, |x_n|\right), \frac{atol}{rtol}\right)} \overset{\text{valid if}}{\leq} \text{rtol} \ .$$

Strategy:

- **Success:** go the the zero-crossing detection step.
- **Failure:** reduce the step-size h_n in general only a division by 2. and restart the integration step with the new step-size.

Remark

The reduction of the step-size is done until the $h_{\min}$ is reached. In that case a simulation error may happen.

Zero-crossing event detection (non adaptive version)

Main steps

- **Detection** of zero-crossing event
  Is one of the zero-crossing changed its sign between $[t_n, t_n + h_n]$?
- **Localization**: if detection is true
  Bracket the most recent zero-crossing time using bisection method.
- **Pass through** the zero-crossing event in two steps:
  - Set the next major output to the left bound of the bracket time.
  - Reset the solver with the state estimate at the right bound of bracket time.

Ingredients for the localization – 1

Linear approximation of the dynamic.

$$\texttt{computeTz}(\sigma_L, \sigma_R) = \min_L \left\{ \sigma_L(t) - x_L \cdot \frac{\sigma_R(t) - \sigma_L(t)}{x_R - x_L} : \forall x \right\}$$

with $\min_L(t) = \min\{t_i \mid t_i \geq \sigma_L(t)\}$.
Remark: σ_L and σ_R represents the environments at the time enclosing the event.

Zero-crossing event detection (non adaptive version)

Main steps

- **Detection** of zero-crossing event
  Is one of the zero-crossing changed its sign between $[t_n, t_n + h_n]$?
- **Localization**: if detection is true
  Bracket the most recent zero-crossing time using bisection method.
- **Pass through** the zero-crossing event in two steps:
  - Set the next major output to the left bound of the bracket time.
  - Reset the solver with the state estimate at the right bound of bracket time.

Ingredients for the localization – 2

Continuous extension (method dependent) to easily estimate state.
For example, `ode23`

$$\begin{aligned}\texttt{interpolX}(\sigma_{t_n}, \sigma_{t_n+h_n}, t) = {} & (2\tau^3 - 3\tau^2 + 1)\sigma_{t_n}(x) + (\tau^3 - 2\tau^2 + \tau)(t_2 - t_1)p_1 \\ & + (-2\tau^3 + 3\tau^2)\sigma_{t_n+h_n}(x) + (\tau^3 - \tau^2)(t_2 - t_1)p_2\end{aligned}$$

with $\tau = \frac{t - t_n}{h_n}$

Update integration step-size

Goals of this step

Performance issue:

- increase the step-size to reduce the number of simulation loop.
- or decrease the step-size to increase the accuracy.

Scheduling issue: must not missed a discrete sampling time.

Main rule

$$\frac{h' = \texttt{changeH}(\sigma(h), \pi) \qquad t' = \min\{\tau \in \mathsf{Eq}(\texttt{sampling}) : \tau > \sigma(t)\}}{\mathsf{Eq}, \pi, \sigma_M \vdash \sigma \xrightarrow{h} \sigma[h \mapsto \min(h', t' - \sigma(t))]}$$

Note: `changeH` increase or decrease h in function of the integration error.

Remark

If a zero-crossing occurred the step-size is left unchanged in regards to the performance issue.

Conclusion

Conclusion and future work

Conclusion

Despite Simulink is a closed source software, **the main algorithms in the solver are known**.

Hope

The definition of a formal semantics will help increasing the confidence in the Simulink design flow with new verification methods.

Future work

- **Pursuing the formalization** of the semantics: reset state, adaptive zero-crossing, integration method like `ode113`, etc.
- **Development of a homemade simulator** of Simulink models to validate experimentally our semantics.
- **Development of a set-based simulator** in order to address the reachability problem of Simulink models.