An Operational Semantics for Simulinks Simulation Engine

LCTES 2012, Beijing, China

Olivier Bouissou, CEA LIST
Alexandre Chapoutot, ENSTA Paris-tech
What is Simulink?

- A graphical language to describe dynamical systems.

\[ \dot{x} = v \]
\[ \dot{v} = \frac{u(t) - b \times v(t)}{m} \]
What is Simulink?

- A graphical language to describe dynamical systems.
- Allows mixing of continuous and discrete evolutions.

\[
\begin{align*}
\dot{x} & = \frac{v}{m} \\
\dot{v} & = \frac{u(t) - b \times v(t)}{m} \\
\end{align*}
\]

\[
u(t) = K_p \ast (v_m - v(\lfloor t \rfloor_k)) + K_i \ast u_i(\lfloor t \rfloor_k)
\]
What is Simulink?

- A graphical language to describe dynamical systems.
- Allows mixing of continuous and discrete evolutions.
- Allows for definition and detection of special non-smooth events.

\[
\begin{align*}
\dot{x} & = \frac{v}{u(t) - b \times v(t)} \\
\dot{v} & = \frac{u(t)}{m}
\end{align*}
\]

\[
u(t) = \begin{cases} 
K_p \ast (v_m - v(\lfloor t \rfloor_k)) + K_i \ast u_i(\lfloor t \rfloor_k) & \text{if } x(t) \leq 7.1 \\
0 & \text{otherwise}
\end{cases}
\]

Olivier Bouissou, CEA LIST Alexandre Chapoutot, ENSTA Paristech
Motivation.

When is simulation used?

<table>
<thead>
<tr>
<th>Short answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Always.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Longer answer.</th>
</tr>
</thead>
<tbody>
<tr>
<td>During the main phases of the development of embedded software:</td>
</tr>
<tr>
<td>- model in the loop</td>
</tr>
<tr>
<td>the control law and the environment are written in Simulink.</td>
</tr>
<tr>
<td>- software in the loop</td>
</tr>
<tr>
<td>the control algorithm is replaced by its implementation in C, and linked with the model of the environment</td>
</tr>
<tr>
<td>- hardware in the loop</td>
</tr>
<tr>
<td>the software is compiled and executed on the embedded architecture, and linked with the model of the environment</td>
</tr>
<tr>
<td>- By many industrial worldwide (over 1.000.000 users of Matlab).</td>
</tr>
</tbody>
</table>
Motivation.

Why should we study simulation?

Models are becoming more and more complex.

- Continuous subsystems with potentially stiff/non-linear dynamics;
- Control-command law implemented using Stateflow or C programs;
- Communication between both worlds is omnipresent.

Consequence: simulation is often the best way to know how a system will behave.

Simulation is imperfect

- Numerical algorithms instead of solving of ODEs;
- Use of floating point numbers;
- Many approximations for the detection of zero-crossing events.

Consequence: we must understand it in order to improve the development process.
Our contribution.

1. A formalization of what Simulink solver does:
   ▶ we highlight the approximations that appear during the simulation process;
   ▶ parametrized by the parameters that the user may change.

2. An operational semantics for Simulink programs containing:
   ▶ continuous blocks;
   ▶ discrete blocks;
   ▶ zero-crossing events;
   ▶ sampling rates.

3. A tool that follows our semantics and conforms with the results of Simulink.
Towards the semantics.

Simulink simulation loop.

1. computes the output value of each block (time $t_n$).
2. stores the input of the unit-delay blocks (for time $t_{n+1}$).
3. computes a new approximation for the integrator blocks (for time $t_{n+1}$).
4. (possibly) treats zero-crossing events.
5. updates step-size and time.
1. computes the output value of each block (time $t_n$).
1. computes the output value of each block (time $t_n$).
2. stores the input of the unit-delay blocks (for time $t_{n+1}$).
1. computes the output value of each block (time $t_n$).
2. stores the input of the unit-delay blocks (for time $t_{n+1}$).
3. computes a new approximation for the integrator blocks (for time $t_{n+1}$).
4. (possibly) treats zero-crossing events.
Towards the semantics.
Simulink simulation loop.

1. computes the output value of each block (time $t_n$).
2. stores the input of the unit-delay blocks (for time $t_{n+1}$).
3. computes a new approximation for the integrator blocks (for time $t_{n+1}$).
4. (possibly) treats zero-crossing events.
5. updates step-size and time
Towards the semantics.

Two special cases.

When updating the value of the integrator blocks, two phenomena may happen.

Rejected step.

- Advanced solvers have a mechanism to estimate the integration error.
- The user may define *tolerances* to make the integration more precise.
- If the solver estimates the error to be bigger than such thresholds, the step is rejected and tried with another step-size.

\[ \frac{1}{m} \]

\[ \frac{1}{s} \]

\[ h = 0.1 \]
When updating the value of the integrator blocks, two phenomena may happen.

**Rejected step.**

- Advanced solvers have a mechanism to estimate the integration error.
- The user may define *tolerances* to make the integration more precise.
- If the solver estimates the error to be bigger than such thresholds, the step is rejected and tried with another step-size.

$$ h = 0.1 $$
Towards the semantics.

Two special cases.

When updating the value of the integrator blocks, two phenomena may happen.

**Rejected step.**

- Advanced solvers have a mechanism to estimate the integration error.
- The user may define *tolerances* to make the integration more precise.
- If the solver estimates the error to be bigger than such thresholds, the step is rejected and tried with another step-size.

$$h = 0.1$$

$$h = 0.05$$
When updating the value of the integrator blocks, two phenomena may happen.

**Rejected step.**

- Advanced solvers have a mechanism to estimate the integration error.
- The user may define *tolerances* to make the integration more precise.
- If the solver estimates the error to be bigger than such thresholds, the step is rejected and tried with another step-size.
When updating the value of the integrator blocks, two phenomena may happen.

**Zero-crossing events.**
- Some blocks can detect special events that occur between two approximation steps.
- This makes the simulation more precise on these events.

\[ h = 0.7 \]
\[ t = 6 \]
Towards the semantics.

Two special cases.

When updating the value of the integrator blocks, two phenomena may happen.

**Zero-crossing events.**

- Some blocks can detect special events that occur between two approximation steps.
- This makes the simulation more precise on these events.

\[
h = 0.7 \\
t = 6
\]
Towards the semantics.

Two special cases.

When updating the value of the integrator blocks, two phenomena may happen.

**Zero-crossing events.**
- Some blocks can detect special events that occur between two approximation steps.
- This makes the simulation more precise on these events.

\[
\begin{align*}
1/m & \\
- b & \\
K_p & \\
h & \\
\end{align*}
\]

\[
\begin{align*}
\frac{1}{s} & \\
\frac{1}{s} & \\
h = 0.7 & \\
t = 6.1324 &
\end{align*}
\]
Towards the semantics.

Variables and states.

**Variables**

- Each block output is a variable $\ell$.
- Each integrator block has a state $x$.
- Each unit-delay block has a state $d$.
- Time and step-size are variables.

\[ \mathcal{V} = \{ \ell_i, x_i, d_i, t, h \} \]

**State of a program**

The state associate to each variable a floating point value.

\[ \sigma : \mathcal{V} \rightarrow \mathbb{R} \]
Towards the semantics.
Equations from a Simulink program.

- We build a set of equations that represent the Simulink program.
- These equations are ordered.

**Equation examples**

\[
\ell_1 + \ell_3 = \ell_2 + \ell_1
\]
Towards the semantics.
Equations from a Simulink program.

- We build a set of equations that represent the Simulink program.
- These equations are ordered.

*Equation examples*

\[ \ell_3 = \ell_2 + \ell_1 \]

\[ \ell_2 = k \times \ell_1 \]
Towards the semantics.
Equations from a Simulink program.

- We build a set of equations that represent the Simulink program.
- These equations are ordered.

**Equation examples**

\[ ℓ_3 = ℓ_2 + ℓ_1 \]
\[ ℓ_2 = k \times ℓ_1 \]

\[ \frac{1}{s} \]

\[ \dot{x}_1 = ℓ_1 \]
Towards the semantics.
Equations from a Simulink program.

- We build a set of equations that represent the Simulink program.
- These equations are ordered.

*Equation examples*

\[ \ell_3 = \ell_2 + \ell_1 \]

\[ \ell_2 = k \times \ell_1 \]

\[ \ell_2 = x_1 \]
\[ \dot{x}_1 = \ell_1 \]

\[ \ell_2 = x_1 \]
\[ d_1 = 0.1 \ell_1 \]
Towards the semantics.
Equations from a Simulink program.

\[
\begin{align*}
\dot{x}_1 &= \ell_1 \\
\dot{x}_2 &= \ell_2 \\
\ddot{d}_1 &= \ell_{11}
\end{align*}
\]

\[\begin{align*}
\ell_2 &= x_1 \\
\ell_3 &= x_2 \\
\ell_{12} &= \ell_1 \\
\ell_8 &= 0 \\
\ell_{15} &= v_m \\
\ell_{14} &= 0.1 \ell_{15} - \ell_{12} \\
\ell_1 &= 1/m \times \ell_4 \\
\ell_4 &= \ell_5 + \ell_6 \\
\ell_5 &= \text{if}(\ell_1 < 7.1; \ell_7; \ell_8) \\
\ell_6 &= -b \times \ell_2 \\
\ell_7 &= \ell_9 + \ell_{10} \\
\ell_9 &= K_i \times \ell_{11} \\
\ell_{10} &= K_p \times \ell_{14} \\
\ell_{11} &= \ell_{12} + \ell_{13} \\
\ell_{13} &= h \times \ell_{14}
\end{align*}\]
Towards the semantics.
Equations from a Simulink program.

We distinguish different sets of equations.

\[
\begin{align*}
\dot{x}_1 &= \ell_1 \\
\dot{x}_2 &= \ell_2 \\
\ddot{d}_1 &= \ell_{11}
\end{align*}
\]

Solver equations

Update equations

| \ell_2 = x_1 | \ell_1 = 1/m \times \ell_4 |
| \ell_3 = x_2 | \ell_4 = \ell_5 + \ell_6 |
| \ell_{12} = d_1 | \ell_5 = \text{if}(\ell_1 < 7.1; \ell_7; \ell_8) |
| \ell_8 = 0 | \ell_6 = -b \times \ell_2 |
| \ell_{15} = v_m | \ell_7 = \ell_9 + \ell_{10} |

\ell_{14} = 0.1 \ell_{15} - \ell_{12}

Minor equations

Minor equations
Overview of the semantics

$$\sigma(t) = \pi(t_{\text{end}})$$

\[\text{Eq, } \pi \vdash \sigma \Rightarrow \sigma\]

\[\sigma(t) < \pi(t_{\text{end}}) \quad \text{Eq, } \pi \vdash \sigma \xrightarrow{M} \sigma_1 \quad \text{Eq, } \pi \vdash \sigma_1 \xrightarrow{u} \sigma_2 \quad \text{Eq, } \pi \vdash \sigma_2 \xrightarrow{s} \sigma'\]

\[\text{Eq, } \pi \vdash \sigma \Rightarrow \sigma'\]

- The semantics mimic the simulation loop.

- Different kinds of transitions:
  - major step transitions;
  - update transitions;
  - solver transitions.
Major steps rules: \( M \rightarrow \)

Goal:
- propagate the (newly updated) value of the states to the whole system;
- compute the output of the system.

How is it done:
- evaluate the major step equations using standard transition rules \( \rightarrow^o \).
- must be careful with the sampling rates.

\[
\begin{align*}
\sigma(t) \not\in S & \quad \Rightarrow \quad \langle \ell := S \ e, \sigma \rangle \rightarrow \sigma \\
\sigma(t) \in S & \quad \Rightarrow \quad \langle \ell := S \ e, \sigma \rangle \rightarrow \sigma[\ell \leftarrow r] \\
\langle e, \sigma \rangle & \rightarrow r \\
\langle e, \sigma \rangle & \rightarrow r \\
\langle \text{Eq}(M), \sigma \rangle & \rightarrow \sigma' \\
\text{Eq, } \pi \vdash \sigma & \xrightarrow{M} \sigma'
\end{align*}
\]
Goal:

- store the current value of the unit-delay blocks for next step.

How is it done:

- simply replace the state associated to the block by the input of the block.
- all the states are changed in parallel.
- must be carefully with the sampling rates.

\[
\begin{align*}
\text{Eq}(d) &= \overline{d} := S \ell \quad \sigma(t) \notin S \\
\text{Eq, } \pi \vdash \sigma \xrightarrow{u} \sigma
\end{align*}
\]

\[
\begin{align*}
\text{Eq}(d) &= \overline{d} := S \ell \quad \sigma(t) \in S \\
\text{Eq, } \pi \vdash \sigma \xrightarrow{u} \sigma[d \mapsto \sigma(\ell)]
\end{align*}
\]
Goal:

- compute the states of the continuous at next step;
- detect zero-crossing events;
- update the step size to increase precision and performance.

How is it done:

- call to numerical integrator to update the states
- use zero-crossing algorithms
- use the step-size transitions

\[
\text{Eq, } \pi, \sigma \vdash \sigma \xrightarrow{i} \sigma_{so} \quad \text{Eq, } \pi, \sigma \vdash \sigma_{so} \xrightarrow{zc} \sigma_{zc} \quad \text{Eq, } \pi \vdash \sigma_{zc} \xrightarrow{h} \sigma' \quad \text{SOLVER} \]

\[\text{Eq, } \pi \vdash \sigma \xrightarrow{s} \sigma'\]
Goal:
▶ compute an approximation of $x_{n+1}$ at $t_n + h_n$;
▶ approximately solve the differential equation $\dot{x} = \ell$.

How is it done:
▶ depending on the solver, we must evaluate the differential equations at various points.
▶ these evaluations are summed up in the Butcher table.
▶ one evaluation means evaluating the minor step equations.

\[ \begin{align*}
\text{\textit{Euler}} & \\
x_{n+1} &= x_n + h_n \times f(x_n) \\
k_1 &= f(t_n, x_n) \\
k_2 &= f(t_n + (1/2)h_n, x_n + (1/2)h_n k_1) \\
k_3 &= f(t_n + (3/4)h_n, x_n + (3/4)h_n k_2) \\
nx_{n+1} &= x_n + h_n ((2/9)k_1 + (1/3)k_2 + (4/9)k_3) \\
k_4 &= f(t_n + h_n, x_{n+1}) \\
zn_{n+1} &= x_n + h_n ((7/24)k_1 + (1/4)k_2 + (1/3)k_3 + (1/8)k_4) \\
\end{align*} \]
Goal:
▶ compute an approximation of $x_{n+1}$ at $t_n + h_n$;
▶ approximately solve the differential equation $\dot{x} = \ell$.

How is it done:
▶ depending on the solver, we must evaluate the differential equations at various points.
▶ these evaluations are summed up in the Butcher table.
▶ one evaluation means evaluating the minor step equations.

\[ \text{Euler} \]
\[
\begin{array}{c|c}
0 & 1 \\
\hline
1 & 1 \\
\end{array}
\]

\[ \text{ODE23} \]
\[
\begin{array}{c|ccc}
0 & 1 & \frac{1}{2} & 0 \\
\hline
\frac{1}{2} & & & \\
\frac{3}{4} & 0 & \frac{3}{4} & 0 \\
1 & \frac{2}{9} & \frac{1}{3} & \frac{4}{9} & 0 \\
\hline
\frac{2}{9} & \frac{1}{3} & \frac{4}{9} & 0 \\
\frac{7}{24} & \frac{1}{4} & \frac{1}{3} & \frac{1}{8} & 0 \\
\end{array}
\]
Goal:
- compute an approximation of $x_{n+1}$ at $t_n + h_n$;
- approximately solve the differential equation $\dot{x} = \ell$.

How is it done:
- depending on the solver, we must evaluate the differential equations at various points.
- these evaluations are summed up in the Butcher table.
- one evaluation means evaluating the minor step equations.
Goal:
- compute an approximation of $x_{n+1}$ at $t_n + h_n$;
- approximately solve the differential equation $\dot{x} = \ell$.

How is it done:
- depending on the solver, we must evaluate the differential equations at various points.
- these evaluations are summed up in the Butcher table.
- one evaluation means evaluating the minor step equations.

$$k_1 = f(x_1)$$
Solver rules.
Integrator rules.

Goal:
- compute an approximation of $x_{n+1}$ at $t_n + h_n$;
- approximately solve the differential equation $\dot{x} = \ell$.

How is it done:
- depending on the solver, we must evaluate the differential equations at various points.
- these evaluations are summed up in the Butcher table.
- one evaluation means evaluating the minor step equations.

\[ k_1 = f(x_1) \]
Goal:

- compute an approximation of $x_{n+1}$ at $t_n + h_n$;
- approximately solve the differential equation $\dot{x} = \ell$.

How is it done:

- depending on the solver, we must evaluate the differential equations at various points.
- these evaluations are summed up in the Butcher table.
- one evaluation means evaluating the minor step equations.

![Diagram](image-url)
Goal:
- compute an approximation of $x_{n+1}$ at $t_n + h_n$;
- approximately solve the differential equation $\dot{x} = \ell$.

How is it done:
- depending on the solver, we must evaluate the differential equations at various points.
- these evaluations are summed up in the Butcher table.
- one evaluation means evaluating the minor step equations.

$$k_1 = f(x_1)$$
$$k_2 = f(x_1 + h/2.k_1)$$
**Goal:**

- compute an approximation of $x_{n+1}$ at $t_n + h_n$;
- approximately solve the differential equation $\dot{x} = \ell$.

**How is it done:**

- depending on the solver, we must evaluate the differential equations at various points.
- these evaluations are summed up in the Butcher table.
- one evaluation means evaluating the minor step equations.

\[ k_1 = f(x_1) \]
\[ k_2 = f(x_1 + h/2.k_1) \]
\[ k_3 = f(x_1 + 3h/4.k_2) \]
Goal:

- compute an approximation of $x_{n+1}$ at $t_n + h_n$;
- approximately solve the differential equation $\dot{x} = \ell$.

How is it done:

- depending on the solver, we must evaluate the differential equations at various points.
- these evaluations are summed up in the Butcher table.
- one evaluation means evaluating the minor step equations.

\[
\begin{align*}
\langle \text{sc}_1(\sigma_M(x), \text{Eq}(x), \pi), \sigma \rangle & \xrightarrow{\circ} \sigma_1 & \langle \text{sc}_2(\sigma_M(x), \text{Eq}(x), \pi), \sigma_1 \rangle & \xrightarrow{\circ} \sigma_2 & \text{check}_\text{err}(\sigma, \sigma_1, \sigma_2, \pi) = 1 \\
\text{Eq}, \pi, \sigma_M \vdash \sigma \xrightarrow{i} \sigma_M[x \mapsto \sigma_1(x), \ h \mapsto \sigma_1(h)] \\
\langle \text{sc}_1(\sigma_M(x), \text{Eq}(x), \pi), \sigma \rangle & \xrightarrow{\circ} \sigma_1 & \langle \text{sc}_2(\sigma_M(x), \text{Eq}(x), \pi), \sigma_1 \rangle & \xrightarrow{\circ} \sigma_2 & \text{check}_\text{err}(\sigma, \sigma_1, \sigma_2, \pi) = 0 \\
\text{Eq}, \pi, \sigma_M \vdash \sigma_M[h \mapsto \max(\pi(h_{\min}), \frac{\sigma(h)}{2})] \xrightarrow{i} \sigma' \\
\text{Eq}, \pi, \sigma_M \vdash \sigma \xrightarrow{i} \sigma'
\end{align*}
\]
Goal:

- detect special events occurring between $t_n$ and $t_{n+1}$;
- enclose these events as soon as possible.

How is it done:

- bisection method to find the first detectable zero-crossing;
- use of dense approximation to avoid calling the solver too often.
**Goal:**

- detect special events occurring between $t_n$ and $t_{n+1}$;
- enclose these events as soon as possible.

**How is it done:**

- bisection method to find the first *detectable* zero-crossing;
- use of *dense approximation* to avoid calling the solver too often.

$$
\Phi(\sigma_1, \sigma_2, t) = (2\tau^3 - 3\tau^2 + 1)\sigma_1(x) + (\tau^3 - 2\tau^2 + \tau)hp_1 + (-2\tau^3 + 3\tau^2)\sigma_2(x) + (\tau^3 - \tau^2)hp_2
$$
Goal:

- detect special events occurring between \( t_n \) and \( t_{n+1} \);
- enclose these events as soon as possible.

How is it done:

- bisection method to find the first \textit{detectable} zero-crossing;
- use of \textit{dense approximation} to avoid calling the solver too often.

\[
\Phi(\sigma_1, \sigma_2, t) = (2\tau^3 - 3\tau^2 + 1)\sigma_1(x) + (\tau^3 - 2\tau^2 + \tau)hp_1 + (-2\tau^3 + 3\tau^2)\sigma_2(x) + (\tau^3 - \tau^2)hp_2
\]
Goal:
- detect special events occurring between $t_n$ and $t_{n+1}$;
- enclose these events as soon as possible.

How is it done:
- bisection method to find the first detectable zero-crossing;
- use of dense approximation to avoid calling the solver too often.

$$\Phi(\sigma_1, \sigma_2, t) = (2\tau^3 - 3\tau^2 + 1)\sigma_1(x) + (\tau^3 - 2\tau^2 + \tau)hp_1 + (-2\tau^3 + 3\tau^2)\sigma_2(x) + (\tau^3 - \tau^2)hp_2$$
Goal:

- detect special events occurring between \( t_n \) and \( t_{n+1} \);
- enclose these events as soon as possible.

How is it done:

- bisection method to find the first detectable zero-crossing;
- use of dense approximation to avoid calling the solver too often.

\[
\begin{align*}
\text{Eq}, \pi \vdash \sigma_{so}[t \mapsto \sigma_M(t) + \sigma_{so}(h)] \xrightarrow{m} \sigma' & \quad \text{dzc}(\sigma_M, \sigma', \text{Eq}) = \text{false} \\
\text{Eq}, \pi, \sigma_M \vdash \sigma_{so} \xrightarrow{zc} \sigma_{so}[t \mapsto \sigma_M(t) + \sigma_{so}(h)] & \quad \text{Eq}, \pi \vdash \sigma_M[t \mapsto \sigma_M(t) + \sigma_{so}(h)] \xrightarrow{m} \sigma' \\
\text{dzc}(\sigma_M, \sigma', \text{Eq}) = \text{true} & \quad \text{Eq}, \pi, \sigma_M, \sigma_{so} \vdash \sigma_M, \sigma' \xrightarrow{\text{loc}} \sigma_L, \sigma_R \quad \text{Eq}, \pi \vdash \sigma_L \xrightarrow{m} \sigma_1 \\
\text{Eq}, \pi \vdash \sigma_1 \xrightarrow{u} \sigma' & \\
\text{Eq}, \pi, \sigma_M \vdash \sigma_{so} \xrightarrow{zc} \sigma'[x \mapsto \sigma_R(x), t \mapsto \sigma_R(t)] & \quad \text{Eq}, \pi, \sigma_M, \sigma_{so} \vdash \sigma_M, \sigma_R \xrightarrow{lr} \sigma'_L, \sigma'_R \quad |\sigma'_L(t) - \sigma'_R(t)| \leq \pi(\epsilon_z) \\
\text{Eq}, \pi, \sigma_M, \sigma_{so} \vdash \sigma_L, \sigma_R \xrightarrow{lr} \sigma'_L, \sigma'_R & \quad |\sigma'_L(t) - \sigma'_R(t)| > \pi(\epsilon_z) \\
\tilde{t} = \text{computeTz}(\sigma_L, \sigma_R) & \quad r = \text{interpolX}(\sigma_M, \sigma_{so}, \tilde{t}) \\
\text{Eq}, \pi \vdash \sigma_L[x \mapsto r, t \mapsto \tilde{t}] \xrightarrow{m} \sigma & \\
\text{Eq}, \pi, \sigma_M, \sigma_{so} \vdash \sigma_L, \sigma_R \xrightarrow{a} \sigma & \quad \text{dzc}(\sigma_L, \sigma, \text{Eq}) = \text{true} \\
\text{Eq}, \pi, \sigma_M, \sigma_{so} \vdash \sigma_L, \sigma_R \xrightarrow{lr} \sigma_L, \sigma & \quad \text{Eq}, \pi, \sigma_M, \sigma_{so} \vdash \sigma_L, \sigma_R \xrightarrow{a} \sigma \quad \text{dzc}(\sigma_L, \sigma, \text{Eq}) = \text{false} \\
\text{Eq}, \pi, \sigma_M, \sigma_{so} \vdash \sigma_L, \sigma_R \xrightarrow{lr} \sigma, \sigma_R
\end{align*}
\]
Goal:
- improve the performance of the solver by adapting the step-size;
- do not miss any sampling times.

How is it done:
- use the estimated error to reach desired tolerance;
- restrict the step to the next sampling time.

\[
h' = \begin{cases} 
    h/\text{temp} & \text{if } \text{err} \leq \text{rtol} \text{ and } \text{temp} > 0.2 \\
    5.0h & \text{if } \text{err} \leq \text{rtol} \text{ and } \text{temp} \leq 0.2 \\
    \max(h_{\min}, 0.5h) & \text{otherwise (decreasing step-size)} 
\end{cases}
\]

with \( \text{temp} = 1.25(\text{err}/\text{rtol})^{1/3} \)

\[
h' = \text{changeH}(\sigma(h), \pi) \\
\begin{aligned}
t' &= \min\{\tau \in \text{Eq(sampling)} : \tau > \sigma(t)\} \\
\text{Eq, } \pi, \sigma_M &\vdash \sigma \xrightarrow{h} \sigma[h \mapsto \min(h', t' - \sigma(t))] 
\end{aligned}
\]
Summing it up.

- Set of 20 inference rules (not counting the evaluation of expressions).
- Captures the main aspects of Simulink language.
- Easy to implement.

- Sufficient to handle more complicated aspects as:
  - enabled subsystems;
  - triggered subsystems.
Is this semantics correct?

- Difficult to define as Simulink’s solver is a black box.
- We believe this semantics is correct because:
  - we defined it by inspecting Simulink debugger;
  - we implemented a tool that parses Simulink programs and follows the rules.
Is this semantics correct?

- Difficult to define as Simulink’s solver is a black box.
- We believe this semantics is correct because:
  - we defined it by inspecting Simulink debugger;
  - we implemented a tool that parses Simulink programs and follows the rules.

Empirical validation of the semantics, we need more (tricky) examples.
An interesting consequence of this work. A bug in Simulink.

We tested our tool on programs with enabled subsystems and a fixed step solver.

We obtained results different from Simulink.

Cause: bug in Simulink which outputs the value computed by the last step in the Butcher table.

Known bug for variable-step solver by Matlab since 2007:

<table>
<thead>
<tr>
<th>Modified</th>
<th>Bug</th>
<th>Product</th>
<th>Status</th>
<th>Watch</th>
</tr>
</thead>
<tbody>
<tr>
<td>15 Mar 2007</td>
<td>361167</td>
<td>Simulink</td>
<td>Fixed in R2007b(7.0)</td>
<td></td>
</tr>
</tbody>
</table>

Summary
Wrong answer bug for models with enabled subsystems and using a variable step solver with zero crossing detection enabled

Description
If a subsystem’s zero crossing is enabled and its States when enabling parameter is set to held, an enabled subsystem will output the last minor timestep when the subsystem is disabled instead of the last major timestep.
Conclusion.

We provided:

- a formal semantics for Simulink simulation engine;
- the whole simulation engine is only 20 inference rules

How can this be used:

- must be extended to deal with other features of Simulink;
- over-approximate the approximation errors that appear during the simulation;
- modify the implementation to perform set based simulation.