Static Analysis of Simulink Programs

Alexandre Chapoutot and Matthieu Martel

CEA LIST
Laboratoire Modélisation et Analyse des Systèmes en Interaction
alexandre.chapoutot@cea.fr

April, 5th 2008
SLA++P’08
Context

Design of embedded control systems:

- Growth of design complexity follows system complexity
- Use of high level languages such as Matlab/Simulink/Stateflow (de facto standard in industry)
Design of embedded control systems:

- Growth of design complexity follows system complexity
- Use of high level languages such as Matlab/Simulink/Stateflow (de facto standard in industry)
- Need formal verification/validation techniques especially for critical systems
- But most techniques are focused on the software
- While the physical environment is often known at the design level (usually described in Simulink programs)
Design of embedded control systems:

- Growth of design complexity follows system complexity
- Use of high level languages such as Matlab/Simulink/Stateflow *(de facto* standard in industry)*
- Need formal verification/validation techniques especially for critical systems
- But most techniques are focused on the software
- While the physical environment is often known at the design level (usually described in Simulink programs)

We present a new static analysis by abstract interpretation of:

- Simulink programs (not code generated from)
- Dealing with physical environment and software
Simulink is:

- A graphical language describing dynamical systems
  - continuous-time systems
  - discrete-time systems
  - A mix of both

- Its main features are:
  - A lot of specific domain libraries (Automotive, Aerospace, ...)
  - Code generator
  - Test case
A short presentation of Simulink

Simulink is:

- A graphical language describing dynamical systems
  - continuous-time systems
  - discrete-time systems
  - A mix of both

- Its main features are:
  - A lot of specific domain libraries (Automotive, Aerospace, ...)
  - Code generator
  - Test case

We focus our work on a subset of pure Simulink language that is

- A dataflow kernel:
  Arithmetic operations
  Continuous-time integrator
  Unit delay
  Switch (conditional statement)
A short presentation of Simulink

Simulink is:

- A graphical language describing dynamical systems
  - continuous-time systems
  - discrete-time systems
  - A mix of both

- Its main features are:
  - A lot of specific domain libraries (Automotive, Aerospace, ...)
  - Code generator
  - Test case

We focus our work on a subset of pure Simulink language that is:

- A dataflow kernel:
  - Arithmetic operations
  - Continuous-time integrator
  - Unit delay
  - Switch (conditional statement)

- No Stateflow programs (state machines)
- No S-functions (functions written in C/C++, Ada or Matlab)
Examples of a Simulink programs - 1

A continuous-time program

A mechanical system: a mass-spring-damper system

\[ F(t) = m\ddot{x}(t) + c\dot{x}(t) + kx(t) \]

- \( m \) is the mass
- \( c \) is the damping coefficient
- \( k \) is the spring constant

Simulink model:
A simple discrete-time program

- Compute the mean of the current input and the previous one. If the result is greater than zero then true else false.
A simple discrete-time program

- Compute the mean of the current input and the previous one. If the result is greater than zero then true else false.

Putting all together:

- May represent a control which switch on/off the stop light in a car.
Goals of the static analysis

Giving a correction criterion that is:
Evaluating distance between mathematics and computer results
  - In the continuous-time part:
    Evaluate the numerical integration behaviors
  - In the discrete-time part:
    Evaluate the floating-point behaviors

Dealing with sets of values:
  - Considering a set of scenarios

So we define a new validation method of Simulink programs named:

Abstract Simulation

based on a synchronous dataflow semantics
Differential equation systems are associated to Simulink continuous time systems

\[
\begin{align*}
    s_1(t) &= ln1(t) \\
    s_2(t) &= s_4(t) + s_5(t) \\
    s_3(t) &= s_{10}(t) \times \frac{1}{m} \\
    Out1(t) &= s_8(t) \\
    s_{10}(t) &= s_1(t) - s_2(t) \\
    s_4(t) &= s_7(t) \times c \\
    s_5(t) &= s_9(t) \times k \\
    \dot{s}_7(t), \dot{s}_6(t) &= s_3(t) \\
    \dot{s}_9(t), \dot{s}_8(t) &= s_6(t)
\end{align*}
\]
Semantic equations
continuous-time systems

Differential equation systems are associated to Simulink continuous time systems

\[
\begin{align*}
    s_1(t) &= \ln 1(t) \\
    s_2(t) &= s_4(t) + s_5(t) \\
    s_3(t) &= s_{10}(t) \times \frac{1}{m} \\
    \text{Out 1}(t) &= s_8(t) \\
    s_{10}(t) &= s_1(t) - s_2(t)
\end{align*}
\]

\[
\begin{align*}
    s_4(t) &= s_7(t) \times c \\
    s_5(t) &= s_9(t) \times k \\
    \dot{s}_7(t), \dot{s}_6(t) &= s_3(t) \\
    \dot{s}_9(t), \dot{s}_8(t) &= s_6(t)
\end{align*}
\]
Numerical integration using Euler algorithm.

\[ \dot{y}(t) = x(t) \]
is transformed into
\[ y(k+1) \approx y(k) + h \ast x(k) \]

\( h \) is the integration step
(here is 0.01)

\[
\begin{align*}
    s_1(k) &= \text{In1}(k) \\
    s_2(k) &= s_4(k) + s_5(k) \\
    s_3(k) &= s_{10}(k) \ast \frac{1}{m} \\
    \text{Out1}(k) &= s_8(k) \\
    s_{10}(k) &= s_1(k) - s_2(k) \\
    s_4(k) &= s_7(k) \ast c \\
    s_5(k) &= \frac{s_9(k)}{m} \\
    s_7(k), s_6(k) &= \text{state}_1(k) \\
    s_9(k), s_8(k) &= \text{state}_2(k) \\
    \text{state}_1(k+1) &= \text{state}_1(k) + 0.01 \ast s_3(k) \\
    \text{state}_2(k+1) &= \text{state}_2(k) + 0.01 \ast s_6(k)
\end{align*}
\]
Correction criterion of continuous-time systems: intuition

For a differential equation systems $S$:
- A numerical solution of $S$ is a sequence of points that are approximated values of the solution of $S$
- Validated numerical solution of $S$ is a sequence of intervals that are guaranteed to enclose real solution of $S$ (based on Taylor method [Nedialkov’99])

The distance between validated numerical solution and the numerical solution gives us a correction criterion
Correction criterion of discrete-time systems

Evaluating rounding errors

- A real $r$ is represented by a floating-point value $f$ and a rounding error $e$ such as:
  \[ r = f + e \]

- Extension of arithmetic operations to deal with those values [Martel’04]

Correction criterion:

- the smaller is the error $e$, the more accurate is the numerical result $f$
Preliminary results
Continuous systems: mass-spring-damper

Output values are bounded in [0, 1.2] for infinite sequences

A periodic input
Output with interval Euler integration
Conclusion

Overview of Abstract Simulation

Current and future works:
- Complete the prototype with validated numerical integration algorithm
- Formalize a control flow analysis: detected when mathematics and execution take different decisions
- etc.