EECS 222: Embedded System Modeling Lecture 9

Rainer Dömer

doemer@uci.edu

The Henry Samueli School of Engineering
Electrical Engineering and Computer Science
University of California, Irvine

Lecture 9: Overview

- SLDL Semantics
- Execution and Simulation Semantics
  - Motivating Examples
- Simulation Semantics
  - Discrete Event Simulation (DES)
  - DES Algorithm for SpecC
- Homework Assignment 4
  - SLDL Model of the Canny Edge Detector

SLDL Semantics

- Essential Concepts in Embedded System Models
  - Behavioral hierarchy
    - Concurrency, state transitions, exception handling
  - Structural hierarchy and connectivity
  - Synchronization and communication
  - Timing
  - ➢ SLDL must support these concepts in syntax and semantics
- Language *semantics* define the *meaning* of constructs
  - Execution semantics (for modeling, simulation, and synthesis)
  - Deterministic vs. non-deterministic behavior
  - Preemptive vs. non-preemptive concurrency
  - Atomic operations
  - Safe synchronization and communication

SLDL Semantics

- Language Semantics are needed for …
  - System designer
    - Description and modeling
  - Electronic Design Automation (EDA) tools
    - Validation (compilation, simulation, estimation)
    - Analysis (verification, property checking)
    - Synthesis (implementation)
  - Documentation and standardization
- Objective
  - ➢ Clearly define the execution semantics of the SLDL
- Requirements and Goals
  - Precision (no ambiguities)
  - Abstraction (no implementation details)
  - Formality (enable formal reasoning)
  - Simplicity (easy understanding)

SLDL Semantics Definition

- Example: SpecC language (SystemC is similar)
  - Documentation
    - Language Reference Manual (LRM)
    ⇒ set of rules written in English (somewhat formal)
    - Abstract simulation algorithm
    ⇒ set of valid implementations (abstract, but not general)
  - Reference implementation
    - SpecC Reference Compiler and Simulator
    ⇒ one instance of a valid implementation (very specific)
    - Compliance test bench
    ⇒ set of specific test cases (specific, but incomplete)
  - Formal execution semantics
    - Time-interval formalism
    ⇒ rule-based formalism (mathematical, but incomplete)
    - Abstract State Machines
    ⇒ fully formal approach (algebraic notation, not easy to understand)

Execution and Simulation Semantics

- Motivating Example 1
  - Given:

```
behavior B1(int x)
{
  void main(void)
  {
    x = 5;
  }
};
```

```
behavior B2(int x)
{
  void main(void)
  {
    x = 6;
  }
};
```

```
behavior B
{
  int x;
  B1 b1(x);
  B2 b2(x);

  void main(void)
  {
    b1;
    b2;
  }
};
```

  - What is the value of x after the execution of B?
  - Answer: x = 6

Execution and Simulation Semantics

- Motivating Example 2
  - Given:

```
behavior B1(int x)
{
  void main(void)
  {
    x = 5;
  }
};
```

```
behavior B2(int x)
{
  void main(void)
  {
    x = 6;
  }
};
```

```
behavior B
{
  int x;
  B1 b1(x);
  B2 b2(x);

  void main(void)
  {
    par{b1; b2;}
  }
};
```

  - What is the value of x after the execution of B?
  - Answer: The model is non-deterministic
    (x may be 5, or 6, or any other value!)

Execution and Simulation Semantics

- Motivating Example 3
  - Given:

```
behavior B1(int x)
{
  void main(void)
  {
    waitfor 10;
    x = 5;
  }
};
```

```
behavior B2(int x)
{
  void main(void)
  {
    x = 6;
  }
};
```

```
behavior B
{
  int x;
  B1 b1(x);
  B2 b2(x);

  void main(void)
  {
    par{b1; b2;}
  }
};
```

  - What is the value of x after the execution of B?
  - Answer: x = 5

Execution and Simulation Semantics

- Motivating Example 4
  - Given:

```
behavior B1(int x)
{
  void main(void)
  {
    waitfor 10;
    x = 5;
  }
};
```

```
behavior B2(int x)
{
  void main(void)
  {
    waitfor 10;
    x = 6;
  }
};
```

```
behavior B
{
  int x;
  B1 b1(x);
  B2 b2(x);

  void main(void)
  {
    par{b1; b2;}
  }
};
```

  - What is the value of x after the execution of B?
  - Answer: The model is non-deterministic
    (x may be 5, or 6, or any other value!)

Execution and Simulation Semantics

- Motivating Example 5
  - Given:

```
behavior B1(
  int x, event e)
{
  void main(void)
  {
    x = 5;
    notify e;
  }
};
```

```
behavior B2(
  int x, event e)
{
  void main(void)
  {
    wait e;
    x = 6;
  }
};
```

```
behavior B
{
  int x;
  event e;
  B1 b1(x,e);
  B2 b2(x,e);

  void main(void)
  {
    par{b1; b2;}
  }
};
```

  - What is the value of x after the execution of B?
  - Answer: x = 6

Execution and Simulation Semantics

- Motivating Example 6
  - Given:

```
behavior B1(
  int x, event e)
{
  void main(void)
  {
    notify e;
    x = 5;
  }
};
```

```
behavior B2(
  int x, event e)
{
  void main(void)
  {
    wait e;
    x = 6;
  }
};
```

```
behavior B
{
  int x;
  event e;
  B1 b1(x,e);
  B2 b2(x,e);

  void main(void)
  {
    par{b1; b2;}
  }
};
```

  - What is the value of x after the execution of B?
  - Answer: x = 6

Execution and Simulation Semantics

- Motivating Example 7
  - Given:

```
behavior B1(
  int x, event e)
{
  void main(void)
  {
    waitfor 10;
    x = 5;
    notify e;
  }
};
```

```
behavior B2(
  int x, event e)
{
  void main(void)
  {
    wait e;
    x = 6;
  }
};
```

```
behavior B
{
  int x;
  event e;
  B1 b1(x,e);
  B2 b2(x,e);

  void main(void)
  {
    par{b1; b2;}
  }
};
```

  - What is the value of x after the execution of B?
  - Answer: x = 6

Execution and Simulation Semantics

- Motivating Example 8
  - Given:

```
behavior B1(
  int x, event e)
{
  void main(void)
  {
    x = 5;
    notify e;
  }
};
```

```
behavior B2(
  int x, event e)
{
  void main(void)
  {
    waitfor 10;
    wait e;
    x = 6;
  }
};
```

```
behavior B
{
  int x;
  event e;
  B1 b1(x,e);
  B2 b2(x,e);

  void main(void)
  {
    par{b1; b2;}
  }
};
```

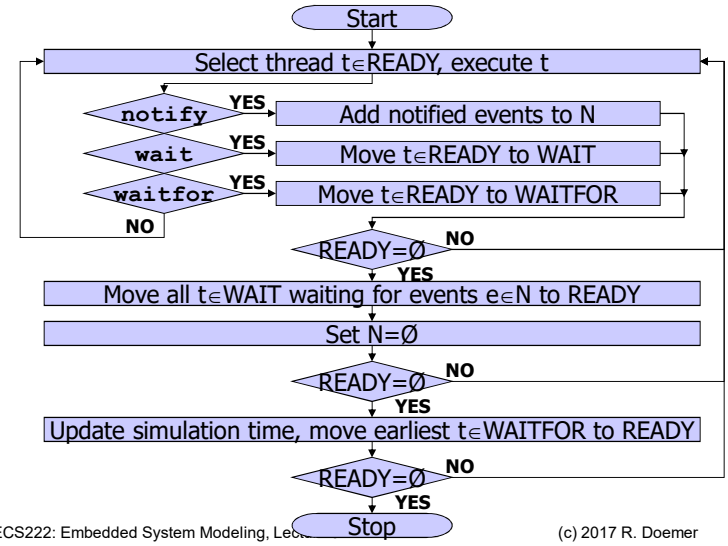
  - What is the value of x after the execution of B?
  - Answer: B never terminates
    (the event is lost!)

Simulation Semantics

- Discrete Event Simulation (DES) Algorithm for SpecC
  - available in LRM (appendix), good for documentation
  - ⇒ abstract definition (defines a set of valid implementations)
  - ⇒ not general (possibly incomplete)
- Definitions:
  - At any time, each thread t is in one of the following sets:
    - **READY**: set of threads ready to execute (initially root thread)
    - **WAIT**: set of threads suspended by `wait` (initially Ø)
    - **WAITFOR**: set of threads suspended by `waitfor` (initially Ø)
  - Notified events are stored in a set **N**
    - `notify e1` adds event `e1` to **N**
    - `wait e1` will wakeup when `e1` is in **N**
    - Consumption of event e means event e is taken out of **N**
    - Expiration of notified events means **N** is set to Ø

Simulation Semantics

- Discrete Event Simulation (DES) Algorithm for SpecC

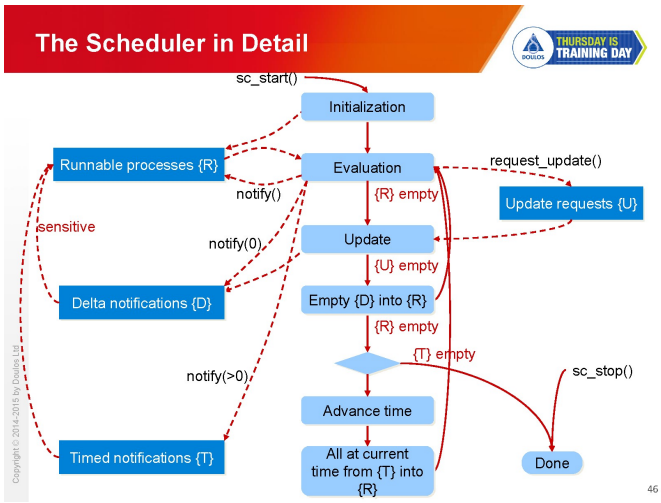


Simulation Semantics

- Discrete Event Simulation (DES) Algorithm for SpecC
  - Conforms to general Discrete Event (DE) Simulation
    - utilizes *delta-cycle* mechanism (i.e. inner event loop)
    - closely matches execution semantics of other languages
      - SystemC
      - VHDL
      - Verilog
  - Features
    - clearly specifies the simulation semantics
    - is easy to understand
    - is straight-forward to implement
  - Generality
    - is one valid implementation of the semantics
    - other valid implementations may exist as well

Simulation Semantics

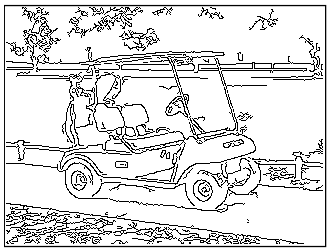
- Discrete Event Simulation (DES) Algorithm for SystemC



EECS 222 Project

- Application Example: Canny Edge Detector
  - Embedded system model for image processing: Automatic Edge Detection in a Digital Camera

golfcart.pgm



golfcart.pgm_s_0.60_l_0.30_h_0.80.pgm

  - Application Source and Documentation:
    - http://marathon.csee.usf.edu/edge/edge_detection.html
    - http://en.wikipedia.org/wiki/Canny_edge_detector

Review: Homework Assignment 1

- Task: Introduction to Application Example
  - Canny Edge Detector
  - Algorithm for edge detection in digital images
- Steps
  1. Setup your Linux programming environment
  2. Download, adjust, and compile the application C code with the GNU C compiler (gcc)
  3. Study the application
- Deliverables
  - None at this time (preparation for following assignments)
- Due
  - By next week: April 10, 2017, 12pm (noon!)

Homework Assignment 4

- Task: SLDL Model of the Canny Edge Detector
  - Convert ANSI-C source code into SLDL model
  - Choose either SpecC or SystemC for simulation
- Steps
  1. Fix the off-by-one bug in the **`non_max_supp`** function
  2. Clean-up the code for compilation without warnings
  3. Fix configuration parameters to compile-time constants
  4. Remove or replace dynamic memory allocation
- Deliverables
  - **`Canny.sc`** or **`Canny.cpp`** (choose one!)
  - **`Canny.txt`**
- Due
  - By next week: May 8, 2017, 12pm (noon!)