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
  - Concepts and Goals
- Execution and Simulation Semantics
  - Motivating Examples (SpecC)
  - Motivating Examples (SystemC)
- Simulation Semantics
  - Discrete Event Simulation (DES)
  - DES Algorithm for SpecC
  - DES Algorithm for SystemC

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

EECS222: Embedded System Modeling, Lecture 9 (c) 2019 R. Doemer 3

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)

EECS222: Embedded System Modeling, Lecture 9 (c) 2019 R. Doemer 4

SLDL Semantics

- Defining Artifacts Available (SpecC and SystemC)
  - 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, SystemC Proof-of-Concept Implementation
    - ⇒ 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 (only exists for SpecC)
    - ⇒ 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 (SpecC)
  - 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 (SpecC)
  - 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!)

EECS222: Embedded System Modeling, Lecture 9 (c) 2019 R. Doemer 7

Execution and Simulation Semantics

- Motivating Example 3 (SpecC)
  - 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

EECS222: Embedded System Modeling, Lecture 9 (c) 2019 R. Doemer 8

Execution and Simulation Semantics

- Motivating Example 4 (SpecC)
  - 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 (SpecC)
  - 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 (SpecC)
  - 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

EECS222: Embedded System Modeling, Lecture 9 (c) 2019 R. Doemer 11

Execution and Simulation Semantics

- Motivating Example 7 (SpecC)
  - 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

EECS222: Embedded System Modeling, Lecture 9 (c) 2019 R. Doemer 12

Execution and Simulation Semantics

- Motivating Example 8 (SpecC)
  - 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!)

Execution and Simulation Semantics

- Motivating Example 9: SystemC Difference
  - Given:

```
SC_MODULE(Top)
{
  int x;

  void th1(void);
  void th2(void);

  SC_CTOR(Top)
  { SC_THREAD(th1);
    SC_THREAD(th2);
  }
};
```

```
void Top::th1(void)
{
  x = 5;
};
```

```
void Top::th2(void)
{
  x = 6;
};
```

  - What is the value of **x** at the end of simulation?
  - Answer: The model is non-deterministic!
    x may have the value 5 or 6,
    but not any other value!

Execution and Simulation Semantics

- Motivating Example 10: SystemC Difference
  - Given:

```
SC_MODULE(Top)
{
  int x;
  sc_event e;
  void th1(void);
  void th2(void);

  SC_CTOR(Top)
  { SC_THREAD(th1);
    SC_THREAD(th2);
  }
};
```

```
void Top::th1(void)
{
  x = 5;
  e.notify();
};
```

```
void Top::th2(void)
{
  wait(e);
  x = 6;
};
```

  - What is the value of **x** at the end of simulation?
  - Answer: *The model is non-deterministic!*
    x *may have the value 5 or 6.*
    The immediate notification may get lost!

EECS222: Embedded System Modeling, Lecture 9 (c) 2019 R. Doemer 15

Execution and Simulation Semantics

- Motivating Example 11: SystemC Difference
  - Given:

```
SC_MODULE(Top)
{
  int x;
  sc_event e;
  void th1(void);
  void th2(void);

  SC_CTOR(Top)
  { SC_THREAD(th1);
    SC_THREAD(th2);
  }
};
```

```
void Top::th1(void)
{
  x = 5;
  e.notify(
     SC_ZERO_TIME);
};
```

```
void Top::th2(void)
{
  wait(e);
  x = 6;
};
```

  - What is the value of **x** at the end of simulation?
  - Answer: **x = 6**
    Delta notification is safe!

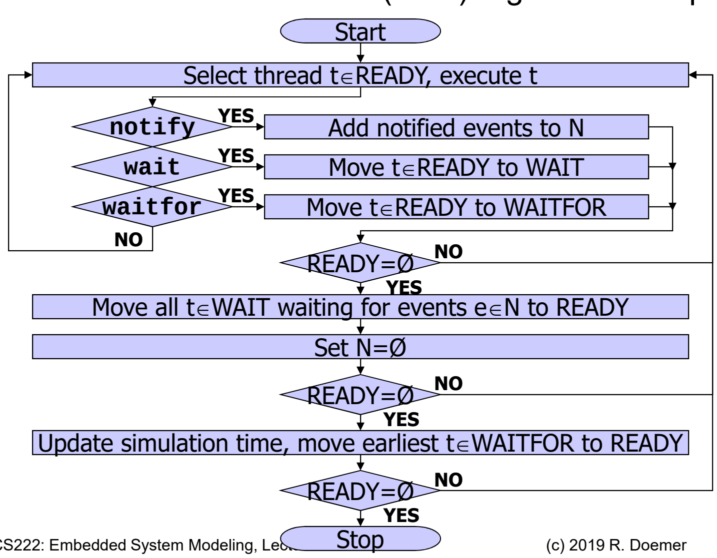
EECS222: Embedded System Modeling, Lecture 9 (c) 2019 R. Doemer 16

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

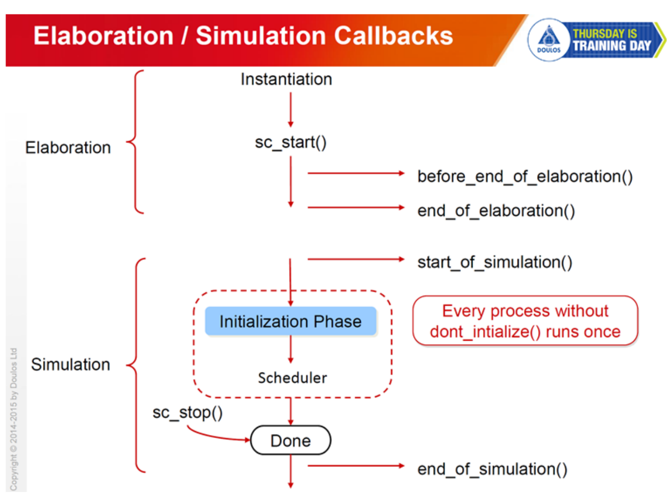


Simulation Semantics

- 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



Simulation Semantics

- Discrete Event Simulation (DES) Algorithm for SystemC