

Process

- May declare fields and functions
- May be extended to define other processes
 - Object orientation a la Java and C++
- Fields and functions are private unless declared public
- Contain at least one constructor and a *thread* function
- Interact with other object through **port**
- **Port**
 - Special field of type *interface*
- **Interface**
 - Declare function with input/output type without implementing the m
- Process access port through calling interface functions

1

An example process

```

process IntX {
  port IntReader port0;
  port IntWriter port1;

  IntX(){}

  void thread() {
    int x;
    while(true) {
      Rd: x=port0.readInt();
      Wr: port1.writeInt(x);
    }
  }
}
    
```

```

interface IntReader extends Port {
  update int readInt();
  eval int n();
}

interface IntWriter extends Port {
  update void writeInt(int data);
  eval int space();
}
    
```

- **Update**
 - Function may change the state of media that implements the interface
- **Eval**
 - Function may NOT change the state of the media...

2

Medium

- Implements interfaces by providing code for the function of the interfaces
- May define fields and functions

3

An example medium

```

Medium IntM implements IntWriter, IntReader, IW, IR, IC, IS, IN {
  int storage, space, n;
  IntM () { space = 1 ; n = 0 ; }
  update void writeInt(int data) {
    await ( space > 0 ; this.IW , this.IS ; this.IW )
    await ( true ; this.IC , this.IS , this.IN ; this.IC ) {
      space = 0 ; n = 1 ; storage = data;
    }
  }
  update int readInt() {
    await ( n > 0 ; this.IR , this.IN ; this.IR )
    await ( true ; this.IC , this.IS , this.IN ; this.IC ) {
      space = 1 ; n = 0 ; return storage;
    }
  }
  eval int space() { await(true;this.IW;this.IC;this.IS) return space;}
  eval int n() { await(true;this.IR;this.IC;this.IN) return n;}
}

interface IW extends Ports {}
interface IR extends Ports {}
interface IC extends Ports {}
interface IS extends Ports {}
interface IN extends Ports {}
    
```

4

await

- The only construct in metamodel for synchronization
- Implement critical sections
- Await (guard ; test list; set list){critical section}
 - Guard: condition that must hold for execution to continue
 - Test list: other process should not have set these
 - Set list: other process can not set these now
- Nondeterminism
 - Await{ (g1 ; t1 ; s1) c1 , (g2 ; t2 ; s2) c2... }

5

Another example process

```

Process Y {
  port IntReader port0;
  port IntReader port1;
  port IntWriter port2;
  ...
  void thread() {
    int z;
    while (true) {
      await {
        (port0.n()>) && port1.n()>0;
        port0.IntReader, port1.IntReader;
        port0.IntReader, port1.IntReader
      } { z = foo(port0.readInt(),port1.readInt()); }
      port2.writeInt(z);
    }
  }
  int foo(int x, int y) {...}
}
    
```

dataflow style

6

Refinement

Define a refinement "pattern":

1. Define objects that constitute the refinement.
2. Define connections among the refinement objects.
3. Specify connections with objects outside the refinement netlist:

Some objects in the refinement may be internally created; others may be given externally.

write a constructor of the refinement netlist for each refinement scenario.

Question: Is it also a behavior equivalence or behavior (trace) refinement?

7

Netlist after Refinement

```

// create mb, and then refine m0 and m1
ByteM mb = new ByteM();
RefIntM refm0 = new RefIntM(m0, mb);
RefIntM refm1 = new RefIntM(m1, mb);
    
```

But, we need coordination:

- if p0 has written to mb, c0 must read
- if p2 has written to mb, c1 must read
- ...

8

Constraints

Two mechanisms are supported to specify constraints:

- 1. Propositions over temporal orders of states**
 - execution is a sequence of states
 - specify constraints using linear temporal logic
 - good for scheduling constraints, e.g.
 - “if process P starts to execute a statement s1, no other process can start the statement until P reaches a statement s2.”
- 2. Propositions over instances of transitions between states**
 - particular transitions in the current execution: called “actions”
 - annotate actions with quantity, such as time, power.
 - specify constraints over actions with respect to the quantities
 - good for real-time constraints, e.g.
 - “any successive actions of starting a statement s1 by process P must take place with at most 10ms interval.”

9

Netlist after Refinement

```

// create mb, and then refine m0 and m1
ByteM mb = new ByteM();
RefIntM refm0 = new RefIntM(m0, mb);
RefIntM refm1 = new RefIntM(m1, mb);
    
```

But, we need coordination:

- if p0 has written to mb, c0 must read
- if p2 has written to mb, c1 must read
- ...

Can be specified using the linear temporal logic.

10

Linear Temporal Logic

```

ltf byteMOrder (IntX p, IntX q, IntX r )
G(end(p, BM.writeByte) -> !beg(q, BM.readByte) U(end(r, BM.readByte));

constraint {
ltf byteMOrder(P0, C1, C0);
ltf byteMOrder(P1, C1, C0);
ltf byteMOrder(P2, C0, C1);
ltf byteMOrder(P3, C0, C1);
}
    
```

11