Metropolis

Metropolis Project Team

in collaboration with

Faculty: Alberto Sangiovanni-Vincentelli
**Metropolis Framework**

**Methodologies**
- Multi-media, wireless communication, mechanical controls, processors

**Meta-model Library**
- Models of computation

**Infrastructure**
- Metropolis meta-model
  - language
  - modeling mechanisms
- Meta-model compiler

**Meta-model Library**
- Architecture platforms

**Tools**
- Simulator
- QSS
- PIG
- STARS
- SPIN
- ...
**Metropolis meta-model**

Concurrent specification with a formal execution semantics:

- **Computation**: $f : X \rightarrow Z$
  - **process**: generates a sequence of *events*

- **Communication**: state enumeration and manipulation
  - **medium**: defines *states* and *methods*

- **Coordination**: constraints over concurrent actions
  - **quantity**: annotated with events
  - **logic**: relates events wrt quantities, defines axioms on quantities
  - **q-manager**: algorithms to realize annotation subject to relations
process P{
    port reader X;
    port writer Y;
    thread(){
        while(true){
            ... 
            z = f(X.read()); 
            Y.write(z);
        }
    }
}

interface reader extends Port{
    update int read();
    eval int n();
}

interface writer extends Port{
    update void write(int i);
    eval int space();
}

medium M implements reader, writer{
    int storage;
    int n, space;
    void write(int z){
        await(space>0; this.writer ; this.writer)
        n=1; space=0; storage=z;
    }
    word read(){ ... }
}
Processes take *actions*.

- statements and some expressions, e.g.
  
  \[ y = z + \text{port.f}();, \quad z + \text{port.f}(), \quad \text{port.f}(), \quad i < 10, \ldots \]

An *execution* of a given netlist is a sequence of vectors of *events*.

- *event*: the beginning of an action, e.g. \( B(\text{port.f}()) \),

  the end of an action, e.g. \( E(\text{port.f}()) \), or null \( N \)

- the \( i \)-th component of a vector is an event of the \( i \)-th process

An execution is *feasible* if

- it satisfies all coordination constraints, and
- it is accepted by all action automata.
Meta-model: action automata

\[ y = x + 1; \]

\[ y := \text{any} \]

\[ x + 1 \]

\[ V_{x+1} := \text{any} \]

\[ B y = x + 1 \]

\[ E x + 1 \]

\[ B x + 1 \]

\[ E x + 1 \]

\[ \ast \]

\[ \ast \]

\[ \ast \]

\[ \ast = \text{write } y \]

\[ \text{write } x \]
Meta-model: execution semantics

- Processes run sequential code concurrently, each at its own arbitrary pace
- Read-Write and Write-Write hazards may cause unpredictable results
  - atomicity has to be explicitly specified
- Progress may block at synchronization points
  - await's
  - function calls and labels to which await-s or LTL constraints refer
Metropolis design environment

- Load designs
- Browse designs
- Relate designs refine, map etc
- Invoke tools
- Analyze results

Metropolis interactive Shell

- Meta model language
- Meta model compiler
- Front end
- Abstract syntax trees
- Back end 1
- Back end 2
- Back end 3
- ... Back end N
- Simulator tool
- Synthesis tool
- Verification tool
**Formal models from the meta-model**

**Example:** Petri nets

```plaintext
await(X.n()>=2; X.reader; X.reader)
for(i=0; i<2; i++) x[i]=X.read();
```

**Restriction:**

Condition inside await is conjunctive.

**Formal methods on Petri nets:**

- analyze the schedulability
- analyze upper bounds of storage sizes
- synthesize schedules