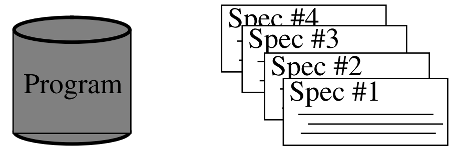
Interfacing System Description Languages to State-Space Exploration-Based Verification

Wendell C. Baker
Electrical Engineering & Computer Science
U.C. Berkeley

December 2, 1993

Design Verification



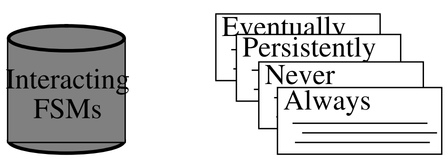
System Model

Problem: *Validate the design database against the given specifications*

- Exhaustive simulation is just not possible even for medium sized designs
- Manual selection of test patterns and manual checking of outputs produced
- Simulation time is a major portion of design time

Want: A formal (mathematical) way to prove for all possible inputs for all possible sequences that the specifications are obeyed

Exhaustive State-Space Exploration



System Model

Method:

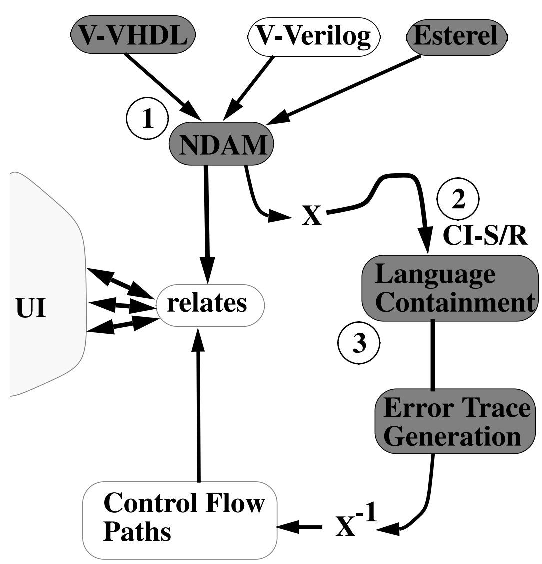
- Examine every state of the system model and determine if any assertion is ever false
- Examine every cycle of the system model and determine if any cycle can fail to perform

Techniques:

Represent the state space and cycles implicitly using Binary Decision Diagrams (BDDs)

Represent the queries as automata and formulate the problem as one of reached-states or cycle-detection

A Verification-Based Debugger



Outline

• Description of Systems

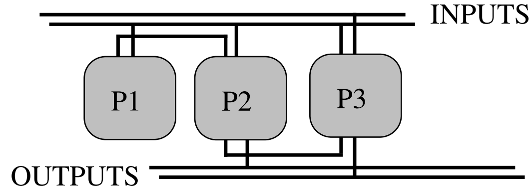
1. Non-Deterministic Abstract Machine (NDAM)

2. The Causally-Interleaved Selection/Resolution Execution Model (CI-S/R)

3. The Language Containment Algorithm

• Conclusions and Future Work

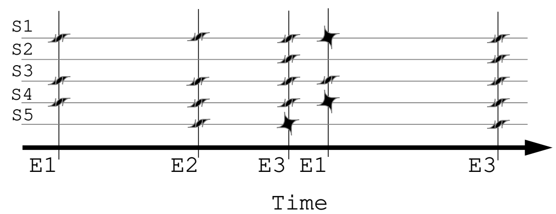
System Description



• A system is a network of reactive processes

• A process communicates using signals

• Time is measured in terms of events on signals



Extensions to the Basic Reactive Process Network

• Subprocesses: want some processes to be activated/deactivated by other processes

An extra *idling* state to suspend child processes

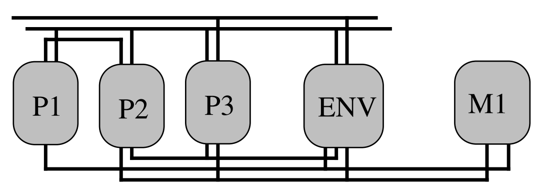
• Preemption: want parent processes to be able to make other processes go idle or restart

Consider that the parent can indicate to indicate to a child that it must go idle

• Exceptions: want a child process to transfer control from itself *and* siblings back the parent

Consider that a child can indicate to the parent and its siblings that the children are going idle and the parent has the control again

Query Description



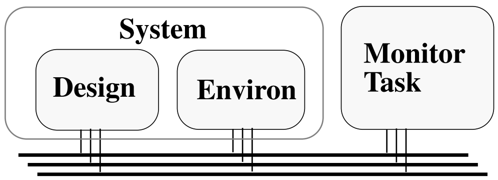
• The system is composed with a description of its environment

• The query is a process that observes the behavior of system-environment pair

• *Assertions* on legal state configurations can specify finite behavior (safety properties)

• *recur edges* and *cycle sets* specify infinite behavior as an L-automaton [Kurshan87] (liveness properties)

Language Containment
[Vardi]



- The complete system executes under the *Selection/Resolution* model of concurrency
- Language containment asks the question

$$L(System) \subseteq L(Task)$$

- Can compute language emptiness

$$L(System) \cap \overline{L(Task)} = \varnothing$$

- If the intersection is non-empty then it represents the set of *error traces* that violate the monitor task

BDDs and Notation
[Bryant86]

Functionally	*Relationally*
$f: B^n \times B^n \rightarrow B^m$	$F: B^n \times B^n \times B^m \rightarrow B$
$\vec{z} = f(\vec{x}, \vec{y})$	$b = F(\vec{x}, \vec{y}, \vec{z})$

- Choose a boolean encoding (bit-level) for the *lhs* and the *rhs* of the relation
- Choose an ordering for the boolean-variables
- Build BDD for the function

$$F(lhs, rhs)$$

Common Operation: [Coudert&Madre89]

$$R_{n+1}(x) = (\exists x, R_n(x) \wedge T(x, y)) \; [y/x]$$

Outline

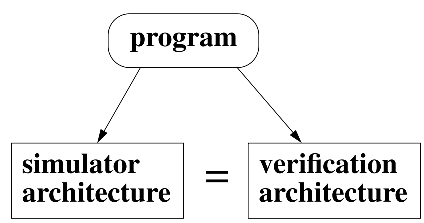
- Description of Systems

---

1. Non-Deterministic Abstract Machine (NDAM)
2. The Causally-Interleaved Selection/Resolution Execution Model (CI-S/R)
3. The Language Containment Algorithm

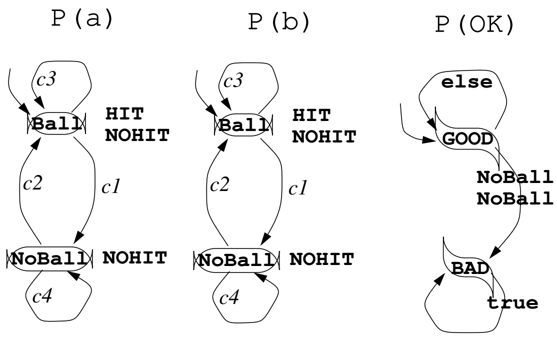
- Conclusions and Future Work

State Transition Operational Semantics



- Simulator architecture choices
  - Computes the interleaving schedule
  - Implement as a table-driven automaton
- Verification architecture choices
  - Compute direct form of the transition relation
  - Use the interleaving implicit in the interactions to simplify the transition relation representation

Example - PingPong



- Two system processes `P(a)` and `P(b)`
  - if `Ball` then either `HIT` it or don't (`NOHIT`)
  - if `NoBall` then `NOHIT`
  - Get `Ball` if if the other guy `Hit` it to you
- One monitor process `P(OK)`
  - if both balls are lost, this is bad

Question: can both balls get lost?

V-VHDL

```
type output_value is (HIT, NOHIT);
signal OA, OB: output_value;
signal tick: bit;

A: process
begin
   loop
     wait on tick; -- the state BALL
     A <= nondetermine HIT, NOHIT;
     if OA'selection = HIT and
             OB'selection = NOHIT then
        exit;
     end if;
   end loop;
   loop
     wait on tick; -- the state NOBALL
     if OB'selection = HIT then
        A <= nondetermine NOHIT;
        exit;
     end if;
   end loop;
   A <= nondetermine NOHIT;
end process A;
```


Non-Deterministic Abstract Machine (NDAM)

The PingPong Example
process `P(a)`

```
L(BALL):   wait on S(tick)
           emit S(a) R(HIT),R(NOHIT)
           R(t1) := selection S(A)
           R(t2) := R(t1) = R(HIT)
           R(t3) := selection S(B)
           R(t4) := R(t3) = R(NOHIT)
           R(t5) := R(t4) and R(t2)
           if not R(t5) goto L(BALL)

L(NOBALL): wait on S(tick)
           R(t6) := selection S(B)
           R(t7) := R(t6) = R(HIT)
           emit S(A) R(NOHIT)
           if R(t7) goto L(BALL)
           goto L(NOBALL)
```


Interesting Instructions

- Signal Read/Write/Reference

```
emit S(a) with R(1), R(2) ...
R(c) := selection R(a)
R(c) := presence R(a)
present S(a) goto L(l)
wait on C(1) S(a)
```

- Nondeterministic Control Flow

```
goto L(1), L(2) ...
```

- Exceptions

```
raise E(e) with R(a)
```

- Subprocess Invocation/Management

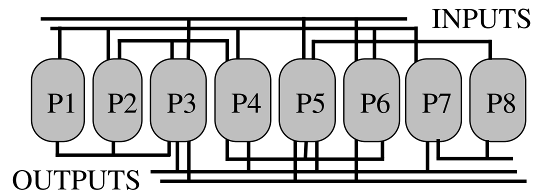
```
try
   P(1) at I(1);
   P(2) at I(2)
watching
   when C(1) S(1) goto L(1) ...
catching
   handle E(1) goto L(1) ...
```

NDAM

- Register-transfer level operations on an infinite-register RISC-like machine
- Features:
  - common register-to-register operations
  - signal read/write and event synchronization
  - goto both deterministic and nondeterministic
  - conditional control flow by if-goto
  - inter-process control flow
- Control-flow and data-flow are separate and fundamentally different
- Generate and optimize with standard code generation techniques

NDAM

The network



- The system is a network of finite-state processes *Pi*
- Each process is reactive: occurrence of an event on an input signal *causes* the process to compute its reaction
- There must be a causal execution ordering among the processes in an instant
- A process must compute a non-infinite amount before suspending to await the next event

Event Derivative Operational Semantics

[Brzozowski64][BerryGonthier88]

- The execution order of processes is governed by the *event derivative* of the network with respect to an input event

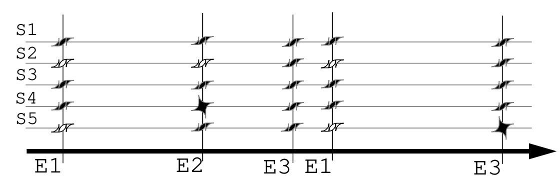
$$\frac{\partial Network}{\partial Event} = P_1, P_2, P_1, P_2, P_3, P_4, P_5, P_6, P_3, P_4, P_5$$

- The event derivative is well-defined only when the processes are *causal* according to the rule
- Once presence (absence) of a signal is decided in an instant, the decision cannot be changed

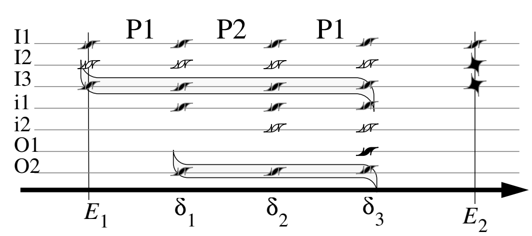
The automata-theoretic interpretation of the NDAM event derivative is Causally-Interleaved Selection/Resolution

Event Derivative

Synchronous: the next-state is described *directly*



Interleaved: the next-state is described *indirectly* as a succession of "deltas"



Single assignment in δ-time

Outline

• Description of Systems

1. Non-Deterministic Abstract Machine (NDAM)

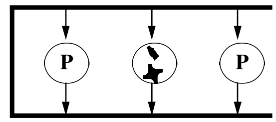
2. The Causally-Interleaved Selection/Resolution Execution Model (CI-S/R)

3. The Language Containment Algorithm

• Conclusions and Future Work

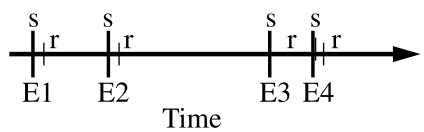
Synchronous Selection/Resolution (S-S/R)

[Kurshan]



• Two-phase execution scheme: (forever)

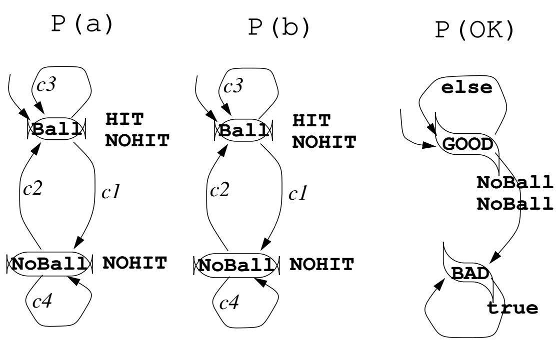
1. selection emit outputs
2. resolution choose next state



• The state-transition behavior of the system is

$$T_{SYS}(x, i, y) = \prod_i T_{P_i}(x, i, y)$$

Example - PingPong

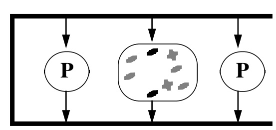


• Two system processes `P(a)` and `P(b)`
- if `Ball` then either `HIT` it or don't (`NOHIT`)
- if `NoBall` then `NOHIT`
- Get `Ball` if if the other guy `Hit` it to you

• One monitor process `P(OK)`
- if both balls are lost, this is bad

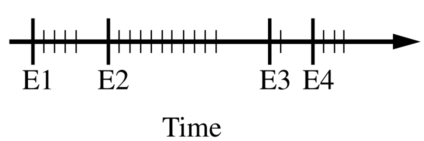
Question: can both balls get lost?

Causally-Interleaved Selection/Resolution (CI-S/R)



• Multi-phase execution scheme: (forever)

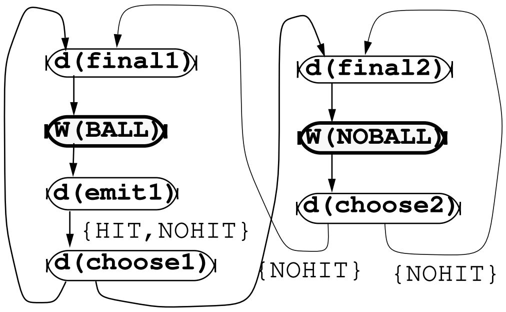
Within each instant; until no change:
- select a little emit some outputs
- resolve a little choose some next states



• The state-transition behavior of the system is

$$\hat{T}_{SYS}(x, i, y) = \sum_i \hat{T}_{P_i}(x, i, y)$$

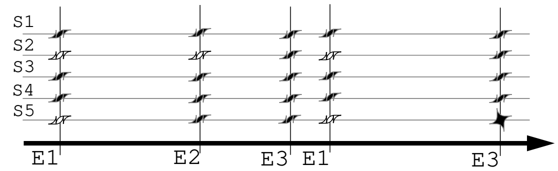
Ping Pong Process `P(a)`



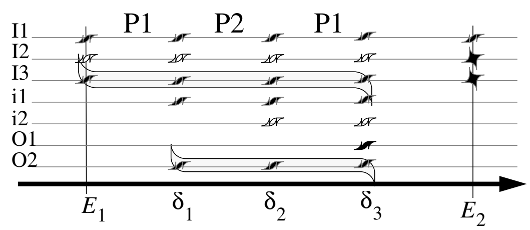
- The `W(`*NAME*`)` are "wait states" and correspond to states of the original synchronous process
- The `d(`*NAME*`)` are "delta states"
- Outputs are associated with "delta transitions" They become defined as the edge is traversed

S-S/R versus CI-S/R

Synchronous: next state *directly* in $T_{SYS}(x, i, y)$



Interleaved: next (macro) state is *indirectly* a succession of deltas states by $\tilde{T}_{P_i}(x, i, y)$



S-S/R versus CI-S/R

Synchronous Execution

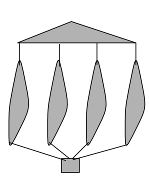
$$T_{SYS}(x, i, y) = \prod_i T_{P_i}(x, i, y)$$

- Each $T_{P_i}(x, i, y)$ describes the transition relation of a process P_i
- The other processes are not mentioned

Interleaved Execution

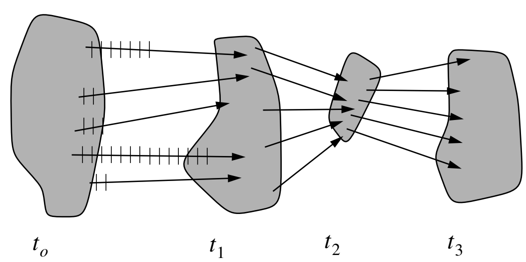
$$\hat{T}_{SYS}(x, i, y) = \sum_i \hat{T}_{P_i}(x, i, y)$$

- Each $\hat{T}_{P_i}(x, i, y)$ describes the transition relation for the process P_i and that the state of the other processes remains stationary



Breadth-First "Simulation"

Forwards / Backwards

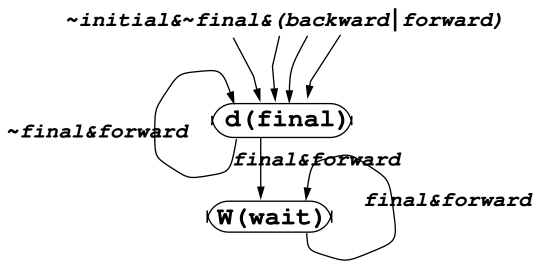


- Represent the set of states of the system at t_i by its characteristic function $Q(x)$ in a BDD
- Represent the transition relation $T(x, i, y)$ also by its characteristic function

$$Q'(x) = (\exists x, Q(x) \wedge T(x, y))\ [y/x]$$
$$Q'(x) = F(Q, T)$$

Problem: for interleaving the number of δ-steps is state-specific: $Q'(x) = \hat{F}^{*}(Q, \hat{T})$

Absorbing End Conditions
(final/forward case)



- Use four extra bits of meta-control

`forward`	simulating forward
`backward`	simulating backward
`initial`	go *wait* to *initial* state
`final`	go *final* to *wait* state

- At each final state create an artificial self-loop to “catch” all of the processes

- Compute $F(Q, \hat{T})$ until a fixed point is reached; then traverse the *final* edge to the wait state

Forward Breadth-first Simulation Algorithm

$$Q'(x) = \hat{F}(Q(x), \hat{T})$$

F(in Q, in T)
return Q1
{
  Qi = *F_initial*(Q, T)
  Qd1 = Qi;
  do {
    Qd = Qd1;
    Qd1 = *F_delta*(Qd, T);
  } while (Qd != Qd1);
  Qf = Qd;
  Q1 = *F_final*(Qf, T);
}

Forward Breadth-first Simulation Components

initial transition

meta-control variables

F_initial(in Qwait, in T)
return Qi
{
  control = initial;
  Qstep1 = Qwait & control;
  do {
    Qstep = Qstep1;
    Qstep1 = *and_smooth_substitute*(
                Qstep, T, x, y);
  } while (Qstep1 != 0);
  Qi = cofactor(Qstep, control);
}

Forward Breadth-first Simulation Components

delta transition

meta-control variables

F_delta(in Qd, in T)
return Qd1
{
  control = ~initial & ~final &
        forward & ~backward;
  Qstep = Qd & control;
  Qstep1 = *and_smooth_substitute*(
        Qstep, T, x, y);
  Qd1 = cofactor(Qstep1, control);
}

Forward Breadth-first Simulation Components

final transition

meta-control variables

F_final(in Qf, in T)
return Qwait
{
  control = final;
  Qstep1 = Qf & control;
  do {
    Qstep = Qstep1;
    Qstep1 = *and_smooth_substitute*(
                Qstep, T, x, y);
  } while (Qstep1 != 0);
  Qwait = cofactor(Qstep, control);
}

Outline

- Description of Systems

1. Non-Deterministic Abstract Machine (NDAM)
2. The Causally-Interleaved Selection/Resolution Execution Model (CI-S/R)

---

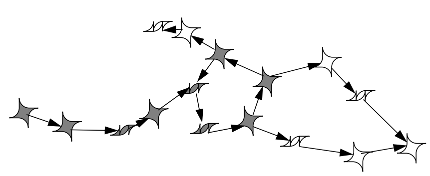
3. The Language Containment Algorithm

- Conclusions and Future Work

Stable Set

[HojatiBrayton92]

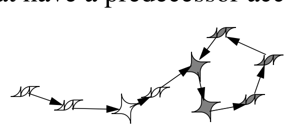
$FSS(Q, T)$ - the Forward Stable Set is the set of all states that have a successor according to T



Greatest fixed point of

$$FS(Q, T) = \exists y, Q(x) \wedge T(x, y)$$

$BSS(Q, T)$ - the Backward Stable Set is the set of all states that have a predecessor according to T



Greatest fixed point of

$$BS(Q, T) = \exists x, Q(y) \wedge T(x, y)$$

L-Automata/L-Processes

[Kurshan87]

$$P = A = (Q, T, I, C, Z)$$

`Q`	the set of states
`T`	the transition relation on Q `T(x,i,y)`: in state `x` on input `i` to `y`
`I`	the start states of the automaton
`C`	cycle sets - a set of sets of states `{ C1, C2, ... Ck }`
`Z`	recur edges - a set of edges in `T`

Language of an L-automaton and L-process can be complementary if deterministic, complete

$L(A)$ stay in a cycle set forever
or traverse a recur edge infinitely

$L(P) = \overline{L(A)}$
do *not* stay in a cycle set forever
and do *not* use a recur edge infinitely

Language Containment (LC) the Synchronous Case

[TouatiBraytonKurshan91][HojatiBrayton92]

1. Compute the system transition relation

$$T_{SYS}(x, y, r) = \prod_i T_{P_i}(x, y, r)$$

2. Compute the reachable states of the system

$$R^*(x) = \mu R, R_0(x) \vee F(R, T)$$

3. Turn “off” the recur edges

$$T^{(r)}(x, y) = T(x, y, r) \wedge (r \equiv 0)$$

4. Compute the states in/between cycles

$$\tilde{Y}(x) = BSS(FSS(R^*, T^{(r)}), T^{(r)})$$

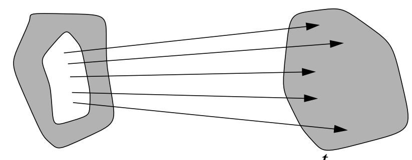
5. Check that all cycles in $\tilde{Y}(x)$ are wholly contained within some cycle set $C_i(x)$

Problem

The LC algorithm depends on a direct-form representation of $T(x, y)$ to in the computation of $FSS(Q, T)$ and $BSS(Q, T)$

$$FSS(Q, T) = \nu Q, FS(Q, T)$$
$$FS(Q, T) = \exists y, Q(x) \wedge T(x, y)$$

Want the set states that have successors in the next instant

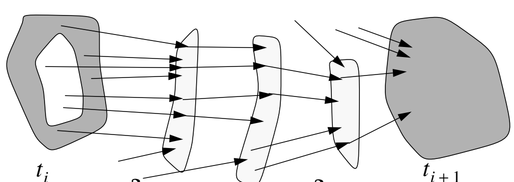


But under Causal Interleaving $\hat{T}(x, y)$ represents only delta transitions.

Transitions between wait-states are represented as fixed-points on $\hat{T}(x, y)$ under a controlled execution scheme

Solution

Use a combination of forward and *restricted* backward simulation to get the same effect



Simulate forward to the next wait state with $\hat{F}(Q, \hat{T})$ and then simulate backwards using a modified $\hat{B}(Q, \hat{T})$

$$FSS(Q, T) = \nu Q, FS(Q, T)$$
$$FS(Q, T) = \hat{F}(\hat{B}(Q, \hat{T}), \hat{T}, H)$$

Where H is a list of the state sets in δ-history used during the $\hat{F}(Q, \hat{T})$ computation

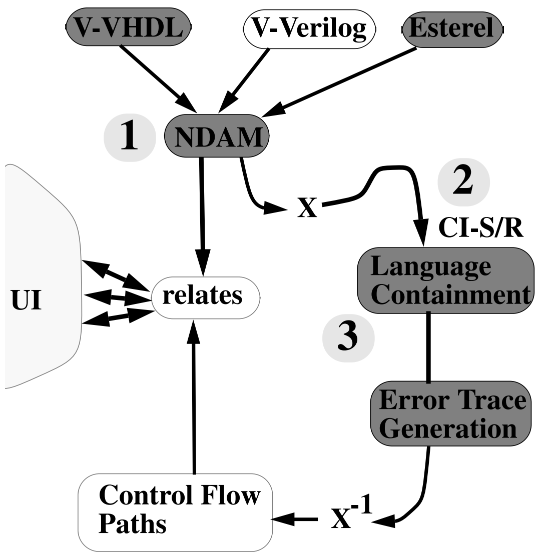
Outline

• Description of Systems

1. Non-Deterministic Abstract Machine (NDAM)
2. The Causally-Interleaved Selection/Resolution Execution Model (CI-S/R)
3. The Language Containment Algorithm

• Conclusions and Future Work

NDAM Intermediate Form



- Sufficiently close to the high-level languages
- Can interpret it in automata-theoretic terms

Conclusions

1. Non-Deterministic Abstract Machine (NDAM)
   - High-level language to automata interface: event-derivative semantics is consistent with Causally Interleaved Selection/Resolution
2. The Causally-Interleaved Selection/Resolution Execution Model (CI-S/R)
   - Makes explicit the fine structure of the next-state behavior that is already present
3. CI-S/R Language Containment Algorithm
   - Use breadth-first simulation with a specially constructed disjunctive transition relation to rederive the fundamental operations

Future Work

- Implement CI-S/R language containment
- Comparison of CI-S/R with S-S/R
  - S-S/R language containment
  - CI-S/R to S-S/R conversion
- Implement NDAM language containment
  - NDAM assembler
  - language-containment
  - error trace generation/recovery
  - NDAM interpreter
- Demonstrate high-level language compilation
  - Synchronous VHDL Compiler
  - Esterel Compiler