Effective Random Testing of Concurrent Programs

What is Concurrency?

- Simplest view
  - Sequential programs: single thread of control
  - Concurrent programs: multiple threads of control

Concurrency Models

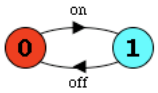
- Many models for studying concurrency
  - These slides use state machines, called Labelled Transition Systems (LTS)
  - Can view and analyze LTS models in using the LTSA analysis tool
- You can download the LTSA tool at:
  - http://www.doc.ic.ac.uk/~jnm/book/ltsa/download.html

Modeling Processes

- Processes - units of sequential execution
  - Model processes as finite state machines
  - Use LTSA to analyse, display and animate behavior

Modeling Processes (cont'd)

- Finite state machines transit from state to state by executing a sequence of atomic actions
- Example: a light switch LTS

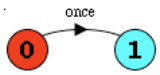


Examining Process Behaviors

- Sequences of actions or *traces*
  - on->off->on->off …
- There can be an infinite number of unique traces

Action Prefix

- If x is an action and P is a process then (x-> P) means that a process does action x and then behaves exactly as described by P
- ONESHOT = (once -> STOP).



- STOP is a special state in which execution logically ends
- Conventions:
  - actions written in lowercase letters
  - PROCESSES written in uppercase letters

Action Prefix & Recursion

- Repetitive behaviour uses recursion:

  SWITCH = OFF,

  OFF = (on -> ON),

  ON = (off-> OFF).
- Substituting gives you:

  SWITCH = OFF,

  OFF = (on ->(off->OFF)).
- And again:

  SWITCH = (on->off->SWITCH).

Choice

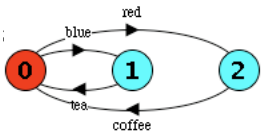
- If x and y are actions then (x-> P | y-> Q) is a process which does action x and then behaves exactly as described by P, or which does action y and then behaves exactly as described by Q

Choice (cont.)

- FSP model of a drinks machine :

```
DRINKS = (red->coffee->DRINKS
                | blue->tea->DRINKS).
```

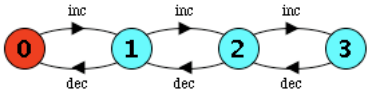
- LTS generated using LTSA:



Guarded Actions

- (when B x -> P | y -> Q) means that when the guard B is true the actions x and y are both eligible to be chosen, otherwise the action x cannot be chosen

```
…
COUNT[i:0..N] =  (when(i<N) inc->COUNT[i+1]
                           |when(i>0) dec->COUNT[i-1]).
```



Modeling Concurrency

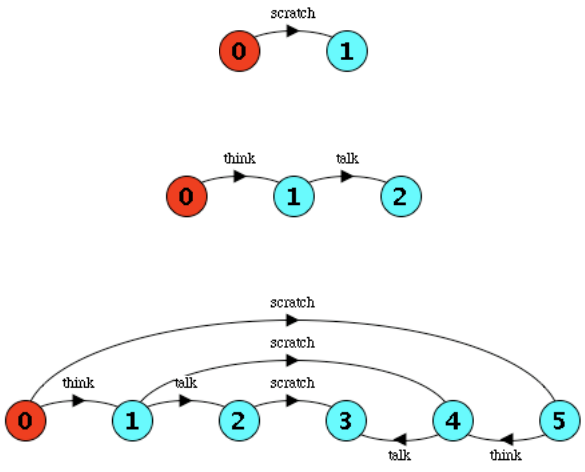
- So far we've modeled individual processes. How do we model concurrent processes?
  - Compose processes by arbitrarily interleaving actions from different processes (but preserve within process order)
- Ignores process execution speed
  - assumes arbitrary speed (abstract away time)

Parallel Composition – Action Interleaving

- If P and Q are processes then (P||Q) represents the concurrent execution of P and Q
- || is the parallel composition operator

ITCH = (scratch->STOP).
CONVERSE = (think->talk->STOP).
||CONVERSE_ITCH = (ITCH || CONVERSE).

ITCH || CONVERSE

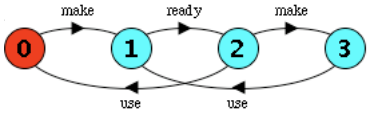


Modeling Interaction - Shared Actions

- If processes in a composition have common actions, those actions are said to be shared
- Shared actions are how process interaction is modeled
- While unshared actions may be arbitrarily interleaved, a shared action must be executed at the same time by all processes that participate in the shared action

MAKER = (make->***ready***->*MAKER)*.
USER = (***ready***->*use->USER)*.
||MAKER_USER = (MAKER || USER).

MAKER||USER



Model Checking Programs

- Situation gets more complicated, but the general ideas are still applicable
- Can model concurrent program executions as interleavings of individual program executions

Example Program & Execution Traces

Initially x = 0 and y = 0

Thread 1:	Thread 2:
y = 1;	if (x==4) assert(false);
y = 2;	
y = 3;	
x = 4;	

Figure 1: Simple Example

```
if (x==4) assert(false); y=1; y=2; y=3; x=4;
y=1; if (x==4) assert(false); y=2; y=3; x=4;
y=1; y=2; if (x==4) assert(false); y=3; x=4;
y=1; y=2; y=3; if (x==4) assert(false); x=4;
y=1; y=2; y=3; x=4; if (x==4) assert(false);
```

Figure 2: Five Interleavings Exhibited by the Simple Example

Observation

- Only variable x is shared between the two threads
- Many of the interleaved executions are therefore equivalent. Only 2 sets of distinct traces:
  - "x=4" in Thread 1 then "if (x==4) …" in Thread 2
  - "if (x==4) …" in Thread 2 then "x=4" in Thread 1
- Creates lots of extra, useless work
- Random exploration oversamples equiv. traces

Partial Order Traces

- If two transitions do not interact with each other then they are independent. All other transitions are dependent. Some examples:
  - Transitions in same thread are dependent
  - Two threads acquiring the same lock is dependent
- Inuitively, we can reorder some independent transitions without affecting the program's behavior

Partial Order Traces (cont.)

- The happens-before relation $\prec$ for a transition sequence $\tau = t_1\ t_2\ \ldots\ t_n$ is defined as the smallest relation such that
  - If t_i and t_j and dependent and $1 \le i \le j \le n$, then $t_i \prec t_j$, and
  - $\prec$ is transitively closed
- All transition sequences that are linearizations of the same partial order are equivalent

RAPOS: Random Partial Order Sampling Algorithm

- Uses two sets of transitions
  - Schedulable: a subset of the transitions that are ready to execute
  - Scheduled: random subset of Schedulable in which all transitions are mutually independent

RAPOS (cont.)

```
RAPOS() {
  s := s0;
  schedulable := Enabled(s);
  while (Enabled(s) ≠ ∅) {
    scheduled := RandIndependentSubset(schedulable);
    for each t ∈ scheduled
      s := Execute(s, t);
    schedulable := {t' ∈ Enabled(s) | ∃t ∈ scheduled
                    such that t and t' are dependent };
    if (schedulable = ∅)
      add a random element from Enabled(s) to schedulable
  }

  if (Alive(s) ≠ ∅) {
    print "Deadlock Detected";
  }
}
```

Figure 4: Random Partial Order Sampling Algorithm (RAPOS)

RAPOS Application

y=1

y=2

y=3

x=4

if (x==4) assert false

schedulable: {y=1} | {if (x==4) assert false} | {y=1, if (x==4) assert false}

RAPOS Application (1st trace)

y=1
y=2
y=3
x=4

if (x==4) assert false

schedulable: {y=1, if (x==4) assert false}
scheduled: {if (x==4) assert false}

RAPOS Application (1st trace)

y=1
y=2
y=3
x=4

if (x==4) assert false

schedulable: {y=1, if (x==4) assert false}
scheduled: {if (x==4) assert false}

RAPOS Application (1st trace)

y=1
y=2
y=3
x=4

if (x==4) assert false

schedulable: {}
scheduled: {if (x==4) assert false}

RAPOS Application (1st trace)

y=1
y=2
y=3
x=4

if (x==4) assert false

schedulable: {y=1}
scheduled: {if (x==4) assert false}

RAPOS Application (1st trace)

y=1
y=2
y=3
x=4

if (x==4) assert false

schedulable: {y=1}
scheduled: {y=1}

RAPOS Application (1st trace)

y=1
y=2
y=3
x=4

if (x==4) assert false

schedulable: {y=1}
scheduled: {y=1}

RAPOS Application (2nd trace)

y=1
y=2
y=3
x=4

if (x==4) assert false

schedulable: {y=1, if (x==4) assert false}
scheduled: {y=1}

RAPOS Application (2nd trace)

y=1
y=2
y=3
x=4

if (x==4) assert false

schedulable: {y=1, if (x==4) assert false}
scheduled: {y=1}

RAPOS Application (2nd trace)

y=1
y=2
y=3
x=4

if (x==4) assert false

schedulable: {y=2}
scheduled: {y=1}

RAPOS Application (2nd trace)

y=1
y=2
y=3
x=4

if (x==4) assert false

schedulable: {y=2}
scheduled: {y=2}