
CIS 721 - Real-Time Systems

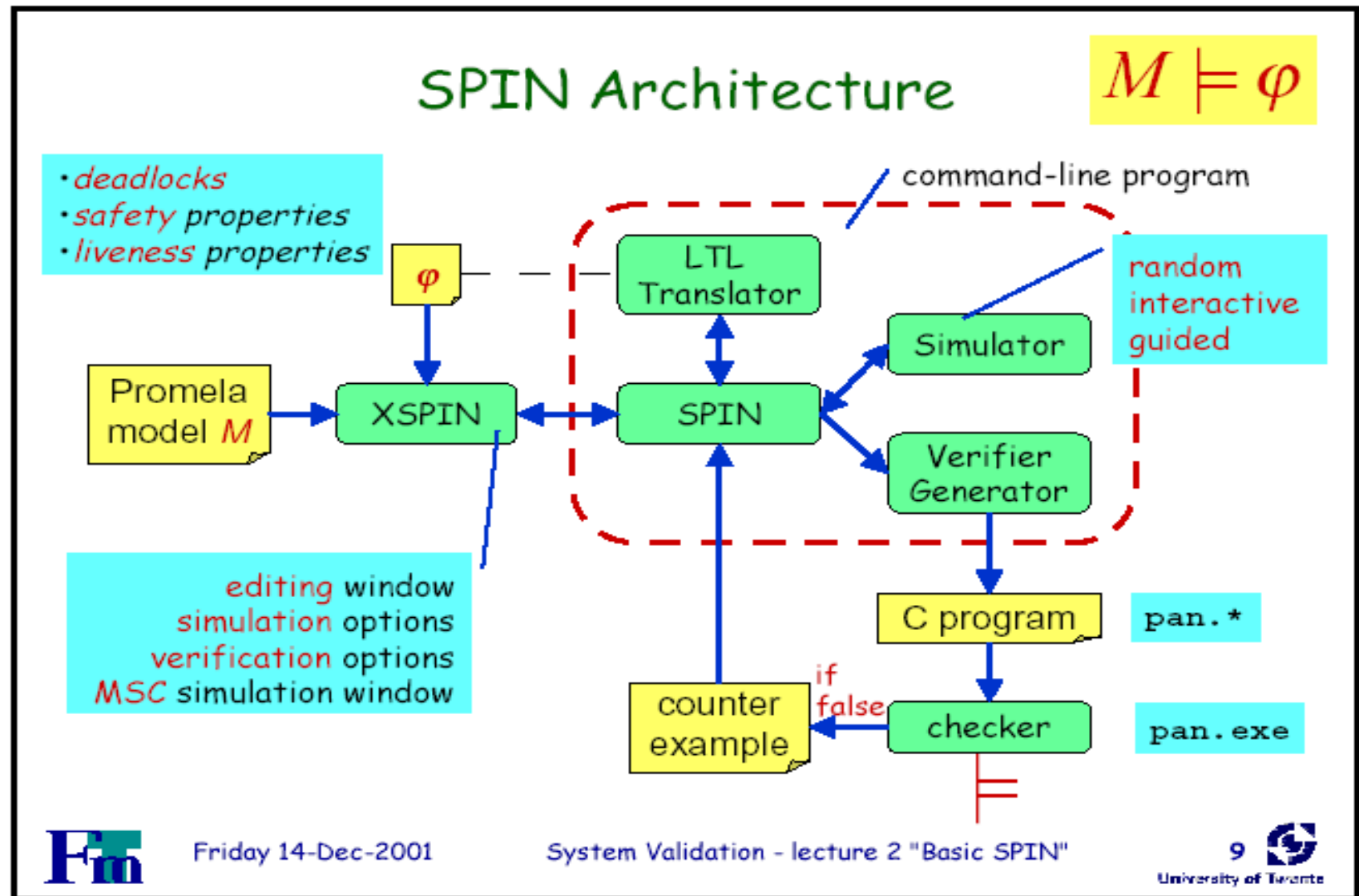
Lecture 33: Advanced SPIN Models

Mitch Neilsen
neilsen@ksu.edu

Uppaal and SPIN Models

- SPIN Summary
- Japanese Raft Puzzle Problem

SPIN Architecture



Concurrency

- SPIN processes execute **concurrently**.
 - Processes are scheduled **non-deterministically**.
 - Processes are interleaved, but statements are executed atomically.
 - Each process may have several different possible actions (statements) enabled at each point in time, but only one action is (non-deterministically) selected to execute.
-

if Statement

- If there is at least one guard (statement) that is executable, then the if statement is executable and SPIN non-deterministically selects one of the executable statements.
- If no guard is executable, then the if statement is blocked (not executable).
- The --> operator is equivalent to $;$. By convention, it is used to separate guards from statements.
- Example:

if

$::$ guard one --> statement a; statement b; statement c;

$::$ guard two --> statement d; statement e; statement f;

fi

Example: Random Number Generators

```
int n;  
if  
  :: skip -> n=1;  
  :: skip -> n=2;  
  :: skip -> n=3;  
fi
```

```
int n;  
if  
  :: n=1;  
  :: n=2;  
  :: n=3;  
fi
```

```
int n;  
select(n : 1..3);
```

do Statement

- With respect to choices, a do statement behaves just like an if statement.
- A do statement simply repeats the choice selection.
- The (always executable) break statement can be used to exit a do-loop.
- Example:

do

 :: guard one -> statement a;

 :: guard two -> statement b; break;

od

Example: Traffic Light

```
mtype = { RED, YELLOW, GREEN };  
active proctype TrafficLight( )  
{  
    byte state = GREEN;  
    do  
        :: (state == GREEN) -> state = YELLOW;  
        :: (state == YELLOW) -> state = RED;  
        :: (state == RED) -> state = GREEN;  
    od;  
}
```


Repetition

```
byte i;
```

```
for (i : 1..10) {  
    /* body of loop */  
}
```

```
i=1;
```

```
do
```

```
:: i > 10 -> goto exitLoop
```

```
:: else -> { /* body of loop */ i++; }
```

```
od;
```

```
exitLoop: printf("out of loop\n");
```

Channels

- Communication between processes is via channels, either for message passing or rendezvous (just set $\langle \text{dim} \rangle = 0$ for a handshake).
- $\text{chan } \langle \text{name} \rangle = [\langle \text{dim} \rangle] \text{ of } \{ \langle \text{type}_1 \rangle, \dots, \langle \text{type}_n \rangle \}$
 - $\langle \text{name} \rangle$ = name of the channel
 - $\langle \text{type}_i \rangle$ = type of elements to be transmitted
 - $\langle \text{dim} \rangle$ = maximum number of elements in the channel
- Example:
 - $\text{mtype} = \{ \text{DATA}, \text{ACK} \};$
 - $\text{chan } c = [5] \text{ of } \{ \text{mtype}, \text{bit} \};$
 - sender executes: $c ! \text{DATA}, 1;$
 - receiver executes: $c ? x, y;$ followed by: $c ! \text{ACK}, y;$

Safety Properties

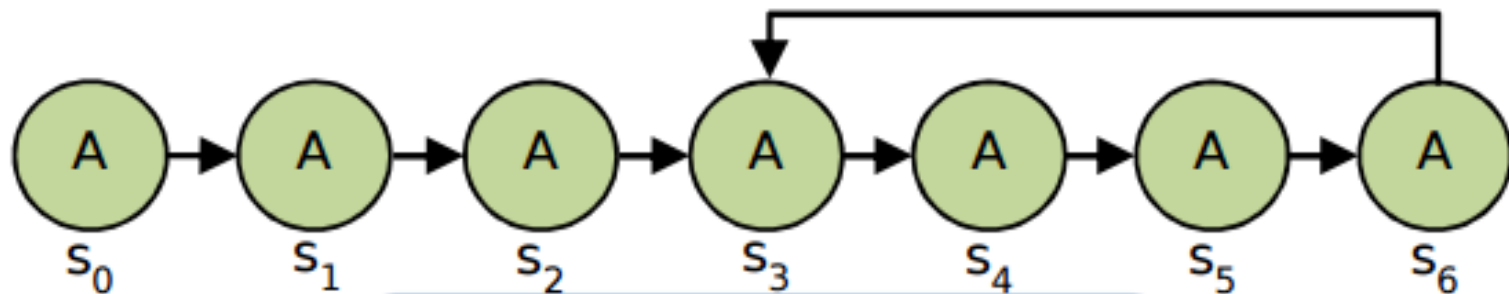
- A **safety property** is used to check if “nothing bad ever happens”.
- **Examples:**
 - **invariants**: x is always less than some constant
 - **deadlock freedom**: the system never reaches a state where no actions are executable
 - **mutual exclusion**: the system never reaches a state where two processes are in the critical section.
- SPIN tries to find a trace leading to the “bad” thing.
- If no such trace exists, then the property is satisfied.

Safety Properties – Verify $[] A$

Safety properties

A counterexample consists of one state where the formula is false. Choose “Safety” in jSpin drop-down menu.

Always A, $[] A$, is true in state s_i if and only if A is true for all states $s_j, j \geq i$.



$[] A$ is true for all states, incl. s_0

Guarded Monitor Process – [] P

- Drawback of solution “1+2 monitor process” is that the **assert** statement is enabled in every state.

```
active proctype monitor( )  
{  
    assert(P) ;  
}
```



```
active proctype monitor( )  
{  
    atomic {  
        !P -> assert(P);  
    }  
}
```

- The **atomic** statement only becomes executable when P itself is **not** true.

Liveness Properties

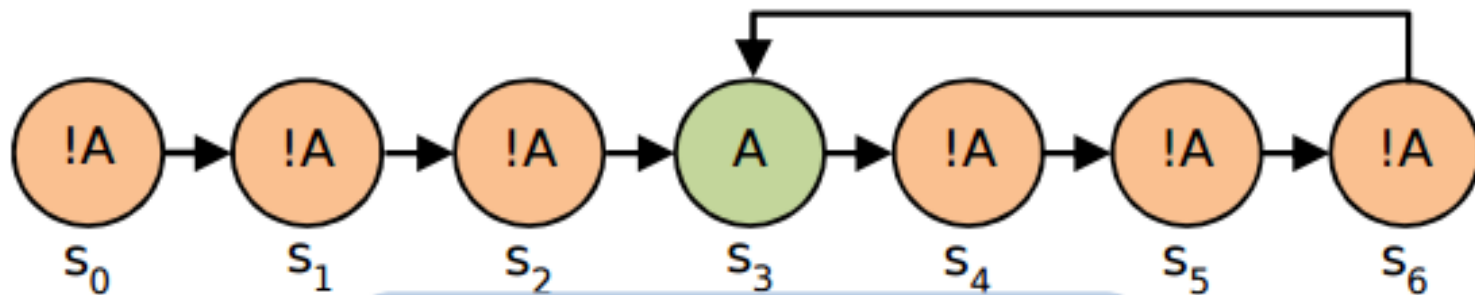
- A **liveness property** is used to check if “something good will eventually happen”.
 - **Examples:**
 - termination: “the system will eventually terminate”
 - response: “if action X occurs, then action Y will occur eventually”
 - SPIN tries to find a (infinite) loop in which the “good” thing does not happen. If there is no such loop, then the property is satisfied.
-

Liveness Property – Eventually \leftrightarrow A

Liveness properties

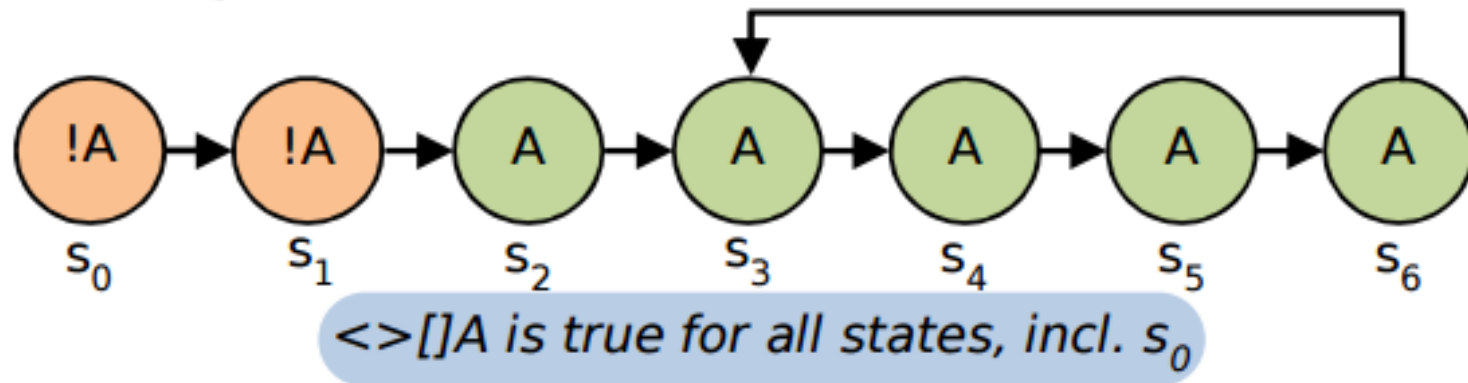
A counterexample is an *infinite* computation where the formula never becomes *true*. Use “Acceptance” in jSpin dropdown menu (and tick of “Weak fairness”).

Eventually A, \leftrightarrow **A**, is true in state s_i if and only if A is true for some state s_j , $j \geq i$.

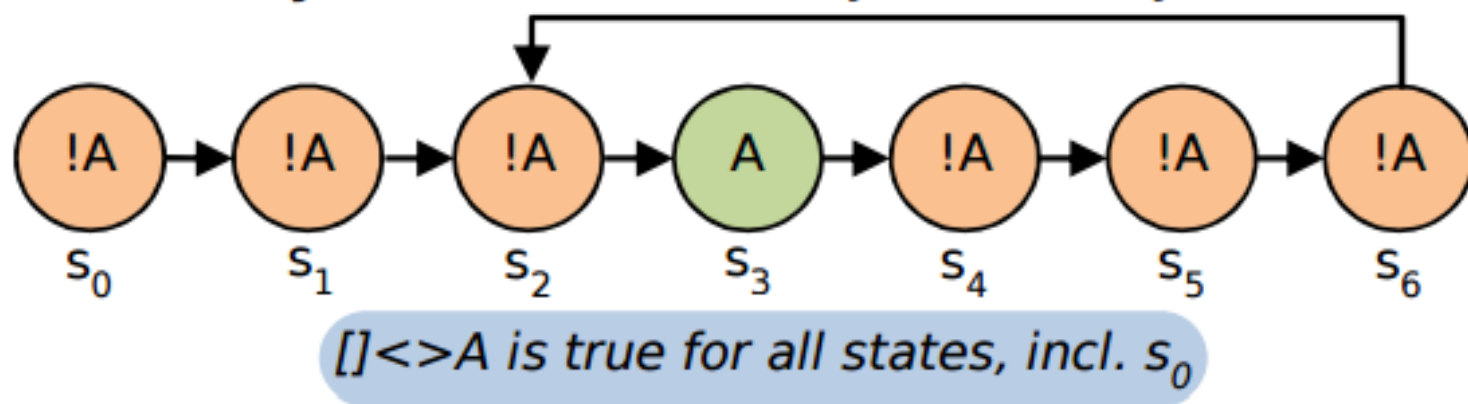


Latching and Indefinitely Often

Latching: $\langle \rangle [A]$ (eventually always)



Indefinitely often: $[\langle \rangle A]$ (always eventually)



Japanese Raft Game

- A group of people must cross a river using a small raft. Unfortunately not everyone gets along and there are certain rules that must be followed in order to get everyone across safely. The group consists of a woman and two girls, a man and two boys, and a policeman with a thief. If you leave certain people alone with others, trouble will ensue. For example, the thief will only behave if the policeman is on the same bank. But the thief can be left alone by himself (just chain him to a log).
- A maximum of two people can be on the raft at a time.
- One adult must be on the raft to operate it; e.g., the woman, man, or policeman.
- The man cannot be with any of the girls without the woman present.
- Conversely, the woman can't stay with the boys without the man there.
- The thief must be with the policeman or be alone.



Chinese Raft Game

- Play game
- Download riverIQGame.swf to play game, and download raft.xml to solve using UPPAAL.



Chinese Puzzle Problem (raft.xml)

C:\uppaal-4.0.10\hwk.xml - UPPAAL

File Edit View Tools Options Help

Editor Simulator Verifier

Drag out

Enabled Transitions

```
movePassengerRight: Man --> Boy1
movePassengerRight: Man --> Boy2
movePassengerRight: Man --> Woman
movePassengerRight: Man --> Girl1
movePassengerRight: Man --> Girl2
movePassengerRight: Man --> Cop
movePassengerRight: Man --> Badguy
movePassengerRight: Man --> Raft
movePassengerRight: Woman --> Man
movePassengerRight: Woman --> Boy1
```

Next Reset

Simulation Trace

```
(left, left, left, left, left, left, left, -, -)
movePassengerRight: Cop --> Badguy
(left, left, left, left, left, left, right, right, -, -)
movePassengerLeft: Cop --> Raft
(left, left, left, left, left, left, left, right, -, -)
movePassengerRight: Cop --> Boy1
(left, right, left, left, left, left, right, right, -, -)
movePassengerLeft: Cop --> Badguy
(left, right, left, left, left, left, left, left, -, -)
movePassengerRight: Man --> Boy2
(right, right, right, left, left, left, left, left, -, -)
movePassengerLeft: Man --> Raft
(left, right, right, left, left, left, left, left, -, -)
movePassengerRight: Man --> Woman
(right, right, right, right, left, left, left, left, -, -)
movePassengerLeft: Woman --> Raft
(right, right, right, left, left, left, left, left, -, -)
```

Trace File:

Prev Next Replay

Open Save Auto

Slow Fast

Man

raft = 0
man = 0
woman = 0
badguy = 0
cop = 0
boy1 = 0
boy2 = 0
girl1 = 0
girl2 = 0

Man

left

raft == 0
movePassengerRight?
man = 1, raft = 1

uhoh?

raft == 0
movePassengerRight!
man = 1, raft = 1

uhoh?

raft == 1
movePassengerLeft!
man = 0, raft = 0

raft == 1
movePassengerLeft?
man = 0, raft = 0

right

Boy1

left

movePassengerRight?
boy1 = 1

movePassengerLeft?
boy1 = 0

right

Boy2

left

movePassengerRight?
boy2 = 1

movePassengerLeft?
boy2 = 0

right

Woman

left

raft == 0
movePassengerRight?
woman = 1, raft = 1

uhoh?

raft == 0
movePassengerRight!
woman = 1, raft = 1

uhoh?

raft == 1
movePassengerLeft!
woman = 0, raft = 0

raft == 1
movePassengerLeft?
woman = 0, raft = 0

right

Girl1

left

movePassengerRight?
girl1 = 1

movePassengerLeft?
girl1 = 0

right

Girl2

left

movePassengerRight?
girl2 = 1

movePassengerLeft?
girl2 = 0

right

Cop

left

raft == 0
movePassengerRight?
cop = 1, raft = 1

uhoh?

raft == 0
movePassengerRight!
cop = 1, raft = 1

uhoh?

raft == 1
movePassengerLeft!
cop = 0, raft = 0

raft == 1
movePassengerLeft?
cop = 0, raft = 0

right

Badguy

left

movePassengerRight?
badguy = 1

movePassengerLeft?
badguy = 0

right

Raft

movePassengerLeft?

movePassengerRight?

Observer

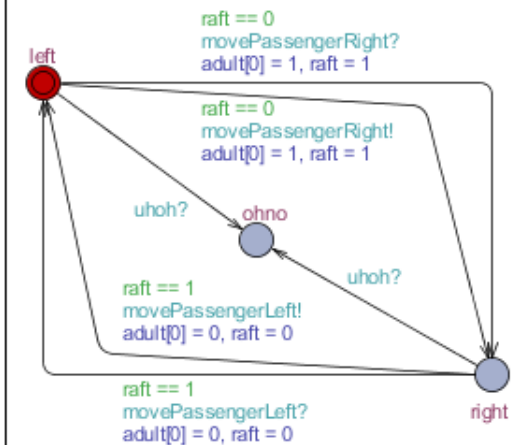
```
((!(woman == boy1) and (woman != man)) or ((woman == boy2) and (woman != man))) or  
((!(man == girl1) and (man != woman)) or ((man == girl2) and (man != woman))) or  
((!(badguy == boy1) and (badguy != cop)) or ((badguy == boy2) and (badguy != cop)) or  
((badguy == girl1) and (badguy != cop)) or ((badguy == girl2) and (badguy != cop)) or  
((badguy == man) and (badguy != cop)) or ((badguy == woman) and (badguy != cop)))
```

uhoh!

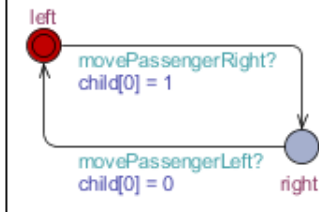
bad

Man Boy1 Boy2 Woman Girl1 Girl2 Cop Badguy Raft Observer

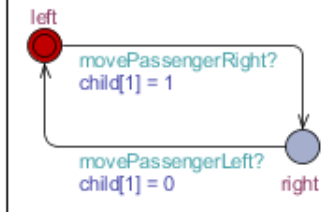
man



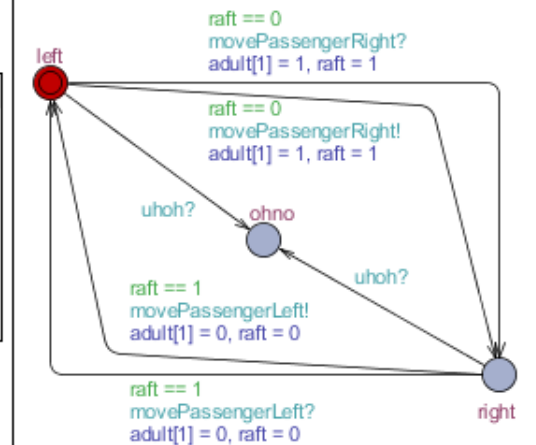
boy1



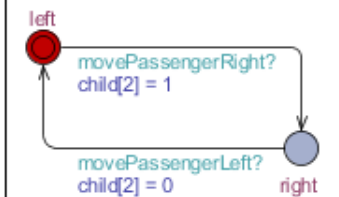
boy2



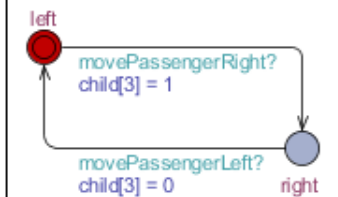
woman



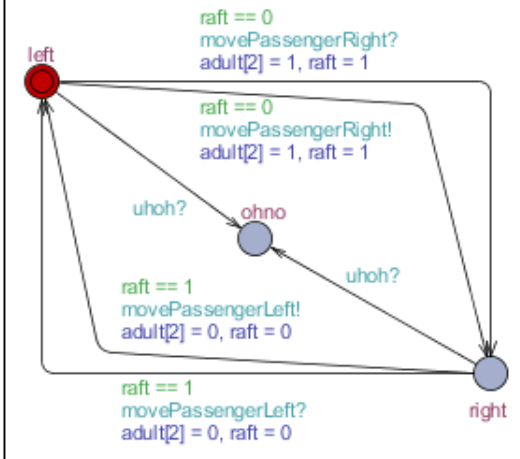
girl1



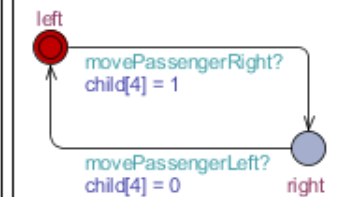
girl2



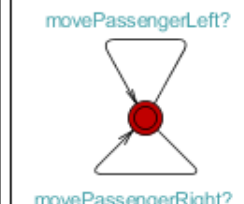
cop



badguy



Raft



Observer

```

(((adult[1] == child[0]) and (adult[0] != adult[1])) or ((adult[1] == child[1]) and (adult[0] != adult[1]))) or
(((adult[0] == child[2]) and (adult[0] != adult[1])) or ((adult[0] == child[3]) and (adult[0] != adult[1]))) or
(((child[4] == child[0]) and (adult[2] != child[4])) or ((child[4] == child[1]) and (adult[2] != child[4]))) or
(((child[4] == child[2]) and (adult[2] != child[4])) or ((child[4] == child[3]) and (adult[2] != child[4]))) or
(((child[4] == adult[0]) and (adult[2] != child[4])) or ((child[4] == adult[1]) and (adult[2] != child[4])))

```

bad

UPPAAL Verification Step

The screenshot displays the UPPAAL software interface. The title bar indicates the file path is `C:\uppaal\raft.xml - UPPAAL`. The menu bar includes `File`, `Edit`, `View`, `Tools`, `Options`, and `Help`. The `Options` menu is open, showing a list of settings: `Search Order`, `State Space Reduction`, `State Space Representation`, `Diagnostic Trace`, `Extrapolation`, `Hash table size`, and `Reuse`. The `Diagnostic Trace` sub-menu is also open, showing options: `None`, `Some`, `Shortest` (selected with a radio button), and `Fastest`. The `Reuse` option has a checkmark. Below the menu, the `Overview` tab is active, showing a query: `E<>{child[0]==1 and child[1]==1 and child[2]==1 and child[3]==1 and child[4]==1 and adult[0]==1 and adult[1]==1 and adult[1]==1 and not Observer.bad}`. The `Query` tab is also visible, showing the same query. The `Comment` tab contains the text: "Does there exist a path on which everyone is safe and the observer has not entered the bad state." The `Status` window at the bottom shows the connection status: "Established direct connection to local server. (Academic) UPPAAL version 4.0.10 (rev. 4417), September 2009 -- server. Disconnected. Established direct connection to local server. (Academic) UPPAAL version 4.0.10 (rev. 4417), September 2009 -- server. E<>(child[0]==1 and child[1]==1 and child[2]==1 and child[3]==1 and child[4]==1 and adult[0]==1 and adult[1]==1 and adult[1]==1 and not Observer.bad) Property is satisfied."

C:\uppaal\raft.xml - UPPAAL

File Edit View Tools Options Help

Editor Simulator Verifier

Overview

E<>{child[0]==1 and child[1]==1 and child[2]==1 and child[3]==1 and child[4]==1 and adult[0]==1 and adult[1]==1 and adult[1]==1 and not Observer.bad}

Options

- Search Order
- State Space Reduction
- State Space Representation
- Diagnostic Trace
 - None
 - Some
 - Shortest
 - Fastest
- Extrapolation
- Hash table size
- Reuse

Query

E<>(child[0]==1 and child[1]==1 and child[2]==1 and child[3]==1 and child[4]==1 and adult[0]==1 and adult[1]==1 and adult[1]==1 and not Observer.bad)

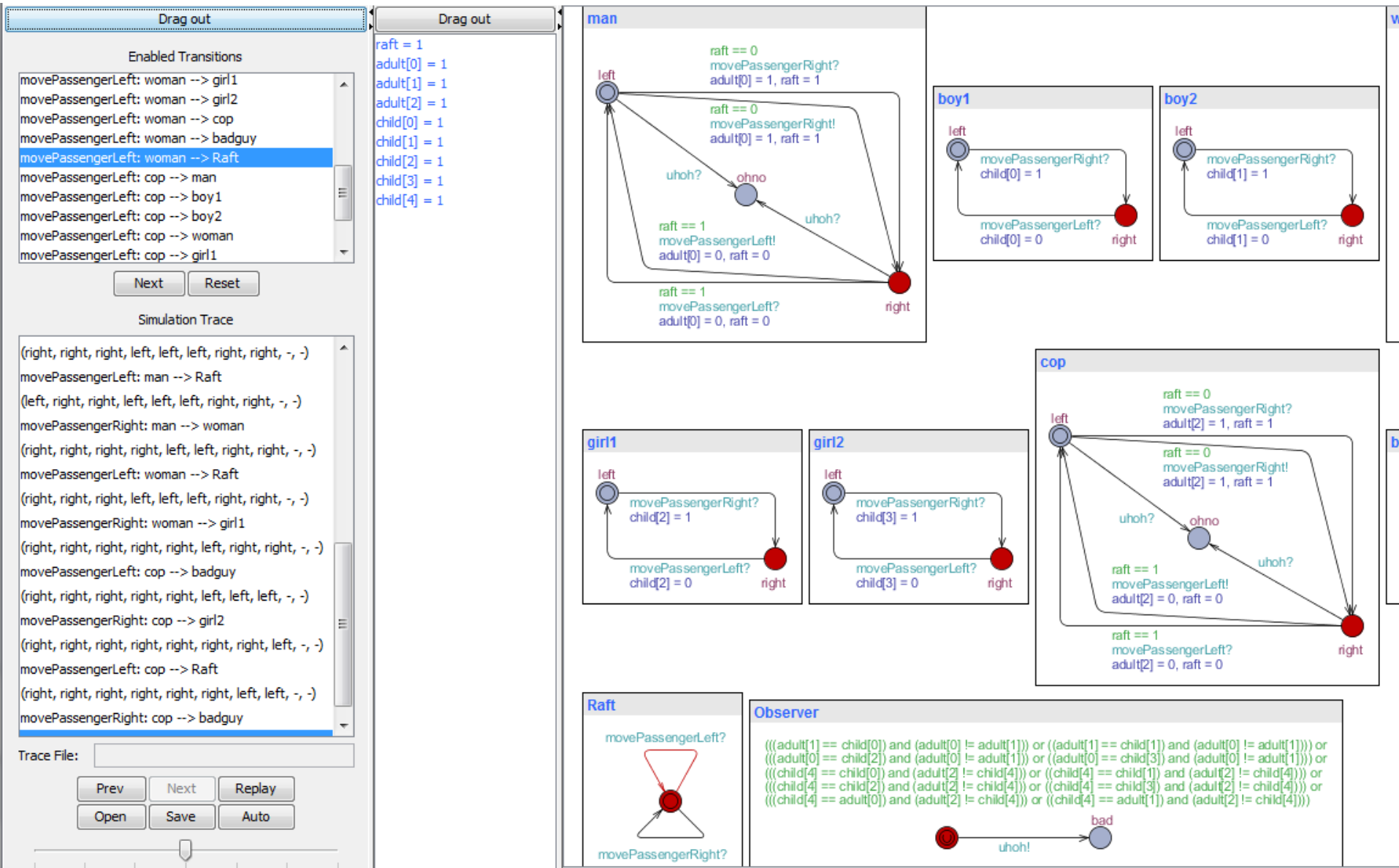
Comment

Does there exist a path on which everyone is safe and the observer has not entered the bad state.

Status

Established direct connection to local server.
(Academic) UPPAAL version 4.0.10 (rev. 4417), September 2009 -- server.
Disconnected.
Established direct connection to local server.
(Academic) UPPAAL version 4.0.10 (rev. 4417), September 2009 -- server.
E<>(child[0]==1 and child[1]==1 and child[2]==1 and child[3]==1 and child[4]==1 and adult[0]==1 and adult[1]==1 and adult[1]==1 and not Observer.bad)
Property is satisfied.

UPPAAL Solution



SPIN Solution Idea

- Model each of the players and allowable moves.
 - If an incorrect move is made, undo the move.
 - Use a Breadth First Search to find the solution involving the fewest number of moves.
-

```

1 mtype = {Cop, Criminal, Mom, Dad, Girl1, Girl2, Boy1, Boy2, Boat};
2
3 #define DONE (r[Cop] == 1 && r[Criminal] == 1 && \
4             r[Mom] == 1 && r[Dad] == 1 && \
5             r[Girl1] == 1 && r[Girl2] == 1 && \
6             r[Boy1] == 1 && r[Boy2] == 1 && r[Boat] == 1)
7
8 #define CriminalBeBad (r[Criminal] != r[Cop] && \
9                     (r[Criminal] == r[Mom] || r[Criminal] == r[Dad] || \
10                    r[Boy1] == r[Criminal] || r[Boy2] == r[Criminal] || \
11                    r[Girl1] == r[Criminal] || r[Girl2] == r[Criminal]))
12 #define BoyNotSafe ( (r[Boy1] == r[Mom] || r[Boy2] == r[Mom]) && r[Mom] != r[Dad] )
13 #define GirlNotSafe ( (r[Girl1] == r[Dad] || r[Girl2] == r[Dad]) && r[Mom] != r[Dad] )
14
15 mtype prev_dr = 0;
16 mtype prev_pass = 0;
17 show mtype driver = 0;
18 show mtype passenger = 0;
19
20 inline printMove(driver, passenger, boat)
21 {
22     if
23     :: boat == 0 ->
24         if
25         :: passenger == 0 -> printf("%e goes across alone.\n", driver);
26         :: else -> printf("%e and %e go across.\n", driver, passenger);
27         fi;
28     :: else ->
29         if
30         :: passenger == 0 -> printf("%e goes back alone.\n", driver);
31         :: else -> printf("%e and %e go back.\n", driver, passenger);
32         fi;
33     fi;
34 }

```



```

36 inline update_r()
37 {
38     r[driver] = 1 - r[driver];
39     if
40     :: passenger != 0 -> r[passenger] = 1 - r[passenger];
41     :: else -> skip;
42     fi;
43     r[Boat] = 1 - r[Boat];
44 }
45
46 inline move(driver, pass)
47 {
48     printMove(driver, pass, r[Boat]);
49     if
50     :: (driver == prev_dr && pass == prev_pass) -> printf("Don't make same move.\n");
51     :: else ->
52         update_r();
53         if
54         :: (CriminalBeBad || BoyNotSafe || GirlNotSafe) -> printf("Bad Move - undo.\n"); update_r();
55         :: else -> prev_dr = driver; prev_pass = pass;
56         fi;
57     fi;
58 }
59
60 /* global array for positions, initially all 0 = not crossed yet */
61 show int r[10];
62 /* mtypes are assigned from 1, arrays are indexed from 0, so r[0] is not used */
63
64 init {
65     do

```

```

64 init {
65     do
66         /* move Cop (with anyone or alone) */
67         :: r[Cop] == r[Boat] -> driver = Cop;
68         if
69             :: r[Criminal] == r[Boat] -> passenger = Criminal
70             :: r[Mom] == r[Boat] -> passenger = Mom
71             :: r[Dad] == r[Boat] -> passenger = Dad
72             :: r[Boy1] == r[Cop] -> passenger = Boy1
73             :: r[Boy2] == r[Cop] -> passenger = Boy2
74             :: r[Girl1] == r[Cop] -> passenger = Girl1
75             :: r[Girl2] == r[Cop] -> passenger = Girl2
76             :: skip -> passenger = 0 /* no passenger */
77         fi;
78         move(driver, passenger);
79         /* move Dad (with a Boy or with Mom or alone) */
80         :: r[Dad] == r[Boat] -> driver = Dad;
81         if
82             :: r[Mom] == r[Boat] -> passenger = Mom
83             :: r[Boy1] == r[Dad] -> passenger = Boy1
84             :: r[Boy2] == r[Dad] -> passenger = Boy2
85             :: skip -> passenger = 0
86         fi;
87         move(driver, passenger);
88         /* move Mom (with a Girl or alone) */
89         :: r[Mom] == r[Boat] -> driver = Mom;
90         if
91             :: r[Girl1] == r[Mom] -> passenger = Girl1
92             :: r[Girl2] == r[Mom] -> passenger = Girl2
93             :: skip -> passenger = 0
94         fi;
95         move(driver, passenger);
96         :: DONE -> printf("SOLVED\n"); assert(0); break;
97         :: else -> printf("WHAT?! \n"); assert(0); break; /* Should never happen! */
98     od;
99 }

```

Verify with default settings...

jSpin Version 5.0

File Edit Spin Convert Options Settings Output SpinSpider Help LTL formula

Open Check Random Interactive Guided Weak fairness ☒ Safety Verify Stop Translate Load LTL name Spin

```
raft3.pml /-
1 mtype = {Cop, Criminal, Mom, Dad, Girl1, Girl2, Boy1, Boy2, Boat};
2
3 #define DONE (r[Cop] == 1 && r[Criminal] == 1 && \
4             r[Mom] == 1 && r[Dad] == 1 && \
5             r[Girl1] == 1 && r[Girl2] == 1 && \
6             r[Boy1] == 1 && r[Boy2] == 1 && r[Boat] == 1)
7
8 #define CriminalBeBad (r[Criminal] != r[Cop] && \
9                     (r[Criminal] == r[Mom] || r[Criminal] == r[Dad] || \
10                    r[Boy1] == r[Criminal] || r[Boy2] == r[Criminal] || \
11                    r[Girl1] == r[Criminal] || r[Girl2] == r[Criminal]))
12 #define BoyNotSafe ( (r[Boy1] == r[Mom] || r[Boy2] == r[Mom]) && r[Mom] == 0)
13 #define GirlNotSafe ( (r[Girl1] == r[Dad] || r[Girl2] == r[Dad]) && r[Dad] == 0)
14
15 mtype prev_dr = 0;
16 mtype prev_pass = 0;
17 show mtype driver = 0;
18 show mtype passenger = 0;
19
20 inline printMove(driver, passenger, boat)
21 {
22     if
23     :: boat == 0 ->
24         if
25         :: passenger == 0 -> printf("%e goes across alone.\n", driver)
26         :: else -> printf("%e and %e go across.\n", driver, passenger)
27     fi;

```

Cop and Criminal go across.
Cop goes back alone.
Cop and Mom go across.
Bad Move - undo.
Cop and Dad go across.
Bad Move - undo.
Cop and Boy1 go across.
Cop and Criminal go back.
Cop and Mom go across.
Bad Move - undo.
Cop and Dad go across.
Bad Move - undo.
Cop and Boy2 go across.
Bad Move - undo.
Cop and Girl1 go across.
Bad Move - undo.
Cop and Girl2 go across.
Bad Move - undo.
Cop goes across alone.
Bad Move - undo.
Dad and Boy2 go across.
Dad and Boy1 go back.
Cop and Criminal go across.

depth-limit (-u250 steps) reached
spin: trail ends after 250 steps
#processes: 1
250: proc 0 (:init:) raft3.pml:65 (state 195)
1 processes created

pan.h:102:2: warning: (near initialization for 'src_file0[2].fnm')
done!
C:\Users\neilsen.n219dw74\classes\cis721\programs\pan -m2000 -X ... done!

Specify Breadth-First Search -DBFS

jSpin Version 5.0

File Edit Spin Convert Options Settings Output SpinSpider Help LTL formula

Open Check Random Interactive Guided Weak fairness ☒ Safety Verify Stop Translate Load LTL n

```
1 - raft3.pml /-
2 mtype = {Cop, Criminal, Mom, Dad, Girl1, Girl2, Boy1, Boy2, Boat};
3
4 #define DONE (r[Cop] == 1 && r[Criminal] == 1 && \
5             r[Mom] == 1 && r[Dad] == 1 && \
6             r[Girl1] == 1 && r[Girl2] == 1 && \
7             r[Boy1] == 1 && r[Boy2] == 1 && r[Boat] == 1)
8
9 #define CriminalBeBad (r[Criminal] != r[Cop] && \
10                    (r[Criminal] == r[Mom] || r[Criminal] == r[Dad] || \
11                    r[Boy1] == r[Criminal] || r[Boy2] == r[Criminal] || \
12                    r[Girl1] == r[Criminal] || r[Girl2] == r[Criminal]))
13 #define BoyNotSafe ( (r[Boy1] == r[Mom] || r[Boy2] == r[Mom]) && r[Mom]
14 #define GirlNotSafe ( (r[Girl1] == r[Dad] || r[Girl2] == r[Dad]) && r
15
16 mtype prev_dr = 0;
17 mtype prev_pass = 0;
18 show mtype driver = 0;
19 show mtype passenger = 0;
20
21 inline printMove(driver, passenger, boat)
22 {
23     if
24     :: boat == 0 ->
25         if
26         :: passenger == 0 -> printf("%e goes across alone.\n", driver)
```

Cop and Criminal go across.
Cop goes back alone.
Cop and Boy1 go across.
Cop and Criminal go back.
Dad and Boy2 go across.
Dad goes back alone.
Dad and Mom go across.
Mom goes back alone.
Cop and Criminal go across.
Dad goes back alone.
Dad and Mom go across.
Mom goes back alone.
Mom and Girl1 go across.
Cop and Criminal go back.
Cop and Girl2 go across.
Cop goes back alone.
Cop and Criminal go across.

Input

C_COMPILER_OPTIONS

-m32 -o pan -DBFS pan.c

OK Cancel

:39 (sta

From the Command Line

- `spin -a raft3.pml`
 - Generate the source file `pan.c` for verification
- `gcc -o pan -DBFS pan.c`
 - Compile the verifier, specify breadth-first-search
- `pan -i`
 - Search for shortest path to error (assertion violation)
 - Will report `Gtk-WARNING **: cannot open display...`
- `spin -t raft3.pml > raft3.trail.out`
 - Generate trail

Spin trail output

cislinux.cis.ksu.edu - PuTTY

```
neilsen@cougar:/pub/cis721$ spin -t raft3.pml
```

```
  Cop and Criminal go across.
```

```
  Cop goes back alone.
```

```
  Cop and Boy1 go across.
```

```
  Cop and Criminal go back.
```

```
  Dad and Boy2 go across.
```

```
  Dad goes back alone.
```

```
  Dad and Mom go across.
```

```
  Mom goes back alone.
```

```
  Cop and Criminal go across.
```

```
  Dad goes back alone.
```

```
  Dad and Mom go across.
```

```
  Mom goes back alone.
```

```
  Mom and Girl1 go across.
```

```
  Cop and Criminal go back.
```

```
  Cop and Girl2 go across.
```

```
  Cop goes back alone.
```

```
  Cop and Criminal go across.
```

```
  SOLVED
```

```
spin: raft3.pml:96, Error: assertion violated
```

```
spin: text of failed assertion: assert(0)
```

```
spin: trail ends after 257 steps
```

```
#processes: 1
```

Spin trail output (cont.)

```
cislinux.cis.ksu.edu - PuTTY
Cop and Criminal go across.
SOLVED
spin: raft3.pml:96, Error: assertion violated
spin: text of failed assertion: assert(0)
spin: trail ends after 257 steps
#processes: 1
    prev_dr = Cop
    prev_pass = Criminal
    driver = Cop
    passenger = Criminal
    r[0] = 0
    r[1] = 1
    r[2] = 1
    r[3] = 1
    r[4] = 1
    r[5] = 1
    r[6] = 1
    r[7] = 1
    r[8] = 1
    r[9] = 1
257:    proc 0 (:init:) raft3.pml:99 (state 198) <valid end state>
1 process created
neilsen@cougar:/pub/cis721$
```



spinroot.com/spin/Man/Pan.html#A



Settings



CollegeNET K-State



HowTo setting up d...



Gmsh: a three-dime...



wyldckat/solidMech...



Microsoft OneDrive

Run-Time Options for *Pan*

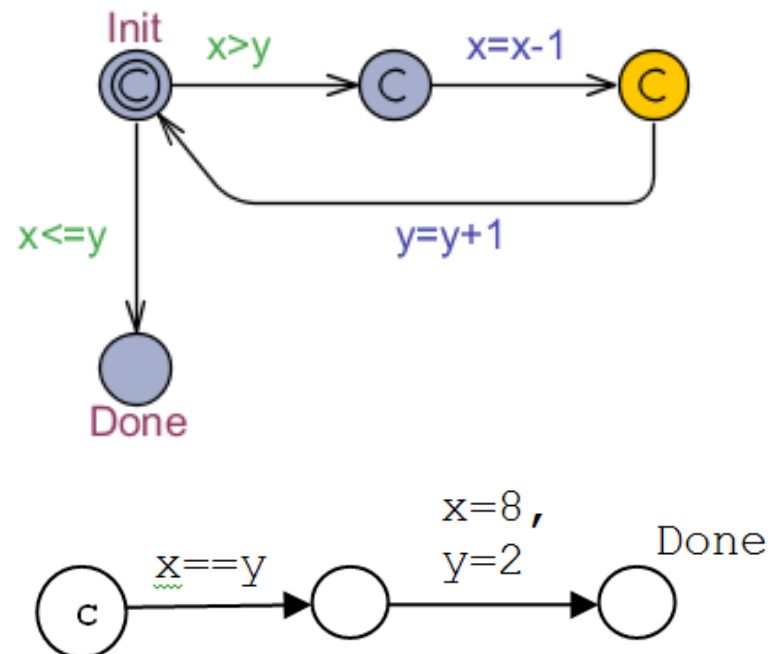
- **-A**
suppress the reporting of assertion violations (see also **-E**)
- **-a**
find acceptance cycles (available if compiled *without* **-DNP**) **-B**
reserved
- **-b**
bounded search mode, makes it an error to exceed the search depth, triggering and error trail
- **-C**
for models with embedded C code, reproduce error trail in columnated format
- **-cN**
stop at *N*th error (defaults to first error if *N* is absent)
- **-d**
print state tables and stop (**-d -d** or **-d -d -d** will print versions of the state tables before additional optimizations are applied)
- **-E**
suppress the reporting of invalid endstate errors (see also **-A**)
- **-e**
create trails for all errors encountered (default is first one only)
- **-Ffilename**
when compiled with **-DSC**, names the file to be used for the stack data
- **-f**
add weak fairness (to **-a** or **-l**)
- **-g**
for models with embedded C code, reproduce error trail with msc gui support
- **-hN**
choose another hash-function, with *N*: 1..32 (defaults to 1)
- **-I**
like **-i**, but approximate and faster
- **-i**
search for shortest path to error (causes an increase of complexity)
- **-J**
reverse the evaluation order of nested unless statements (to conform to the one used in Java)

Promela Model Question

Consider the following Promela model:

```
int x,y;
proctype p() {
  do
    :: (x > y) -> x=x-1; y=y+1;
    :: else -> break;
  od;
}
proctype q() {
  (x == y) ;
  atomic { x=8; y=2};
}
init {
  atomic { x=0; y=0; run p(); run q(); }
}
```

- a) Proctype **p()** can be modeled in UPAAL using the following template, draw an equivalent UPAAL template for proctype **q()**, mark the initial state **Init**, and the final state **Done**.



Promela Model Question

- (b) Will the processes terminate; e.g., is the property $A \Diamond (\underline{p}.\text{Done} \text{ and } \underline{q}.\text{Done})$ satisfied? Explain briefly.
- (c) What are the possible final values for x and y ?
- (d) What if the atomic statement is removed around $\{ \underline{x}=8; \underline{y}=2; \}$? Does your model for proctype $q()$ need to be changed? If so, draw the new model below. Also, in the following list, circle all of the possible final values for x and y :
- $x=8, y=2$
 - $x=8, y=0$
 - $x=4, y=5$
 - $x=5, y=5$
 - $x=4, y=2$
 - $x=3, y=3$

Solution

- (b) Will the processes terminate; e.g., is the property $A \langle \rangle (\underline{p.Done} \text{ and } \underline{q.Done})$ satisfied? Explain briefly. **Yes**
- (c) What are the possible final values for x and y ? **$x=5, y=5$, or $x=8, y=2$**
- (d) What if the atomic statement is removed around $\{ \underline{x=8} ; y=2 ; \}$? Does your model for prototype $q()$ need to be changed? If so, draw the new model below. Also, in the following list, circle all of the possible final values for x and y : **Yes, add another state and transition, the existing transition for $x=8$, and the new next transition for $y=2$.**
- **$x=8, y=2$**
 - **$x=8, y=0$**
 - **$x=4, y=5$**
 - **$x=5, y=5$**
 - **$x=4, y=2$**
 - **$x=3, y=3$**

Summary

- SPIN
 - www.spinroot.com – great source for tutorials, etc.
 - Homework #5 – due Friday, Nov. 20.
 - Quiz #2 – Wed., Dec. 2, in class
 - Open book, open notes
 - Review on Wednesday
-