CIS 721 - Real-Time Systems

Lecture 26: Advanced SPIN

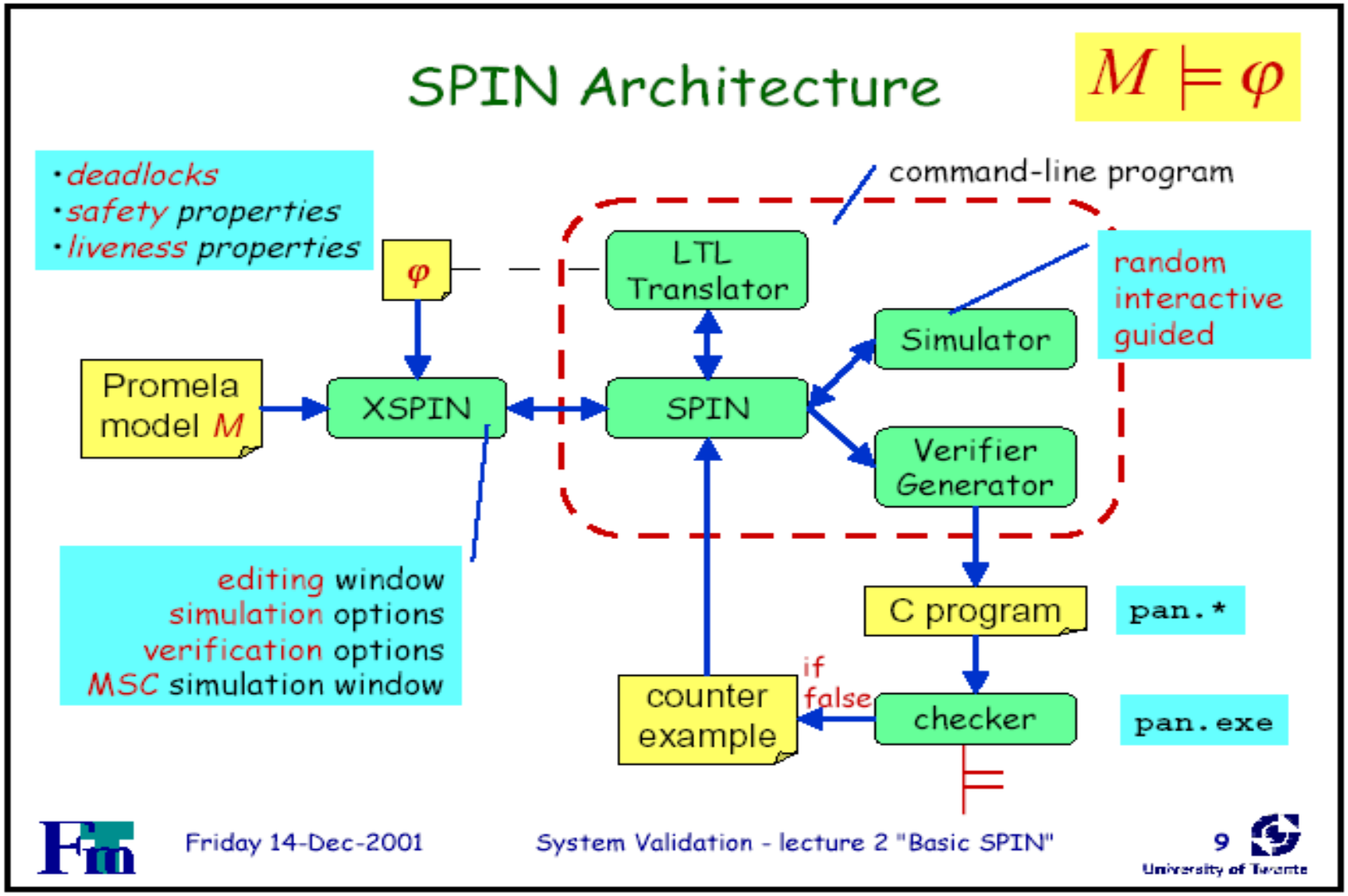
Mitch Neilsen
neilsen@ksu.edu

* some slides from Logic Model Checking, by Mihai Florian, available at spinroot.com or via *www.csee.usf.edu/~zheng/ref/logic-model-checking.pdf*

Outline

- SPIN
  - Advanced topics = SPIN nuances
  - Models
- UPPAAL models

SPIN Architecture



if Statement

- If there is at least one guard (statement) that is executable, then the if statement is executable and SPIN non-deterministically selects on 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 Generator

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

the predefined *expression* else

where in C one writes:

```
if (x <= y) {
    x = y-x;
}
y++;
```

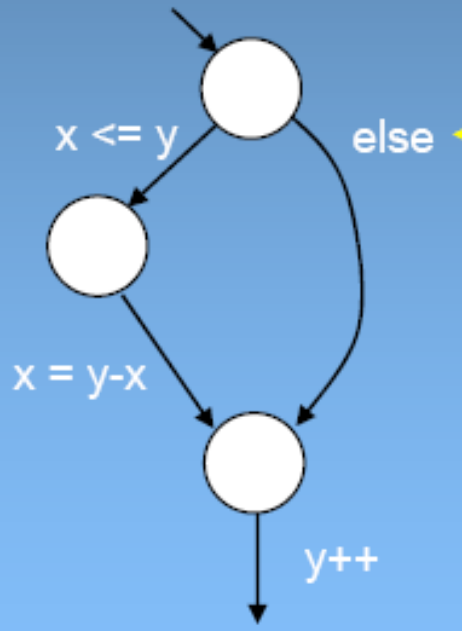
i.e., with implied 'else'

in Promela the 'else' is always explicit:

```
if
:: (x <= y) -> x = y-x
:: else
fi;
y++
```

i.e., the 'else' part cannot be omitted

no need for "-> skip"



in this case 'else' evaluates to:
!(x <= y)

the **else** clause has to be present

without it, the if- statement would block until (x<=y) becomes true

timeout and *else*

- timeout and else are related
  - both are predefined Booleans
  - they evaluate to *true* or *false*, depending on context
  - `else` is *true* iff
    - no other statement in ***the same process*** is executable
  - `timeout` is *true* iff
    - no other statement in ***any process*** is executable
- *timeout* is like a system level *else*, but
  - *else* cannot be combined with other conditionals
  - *timeout* can be combined, e.g. as in (timeout && a > b)

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
```

do-statement

the underlying automaton

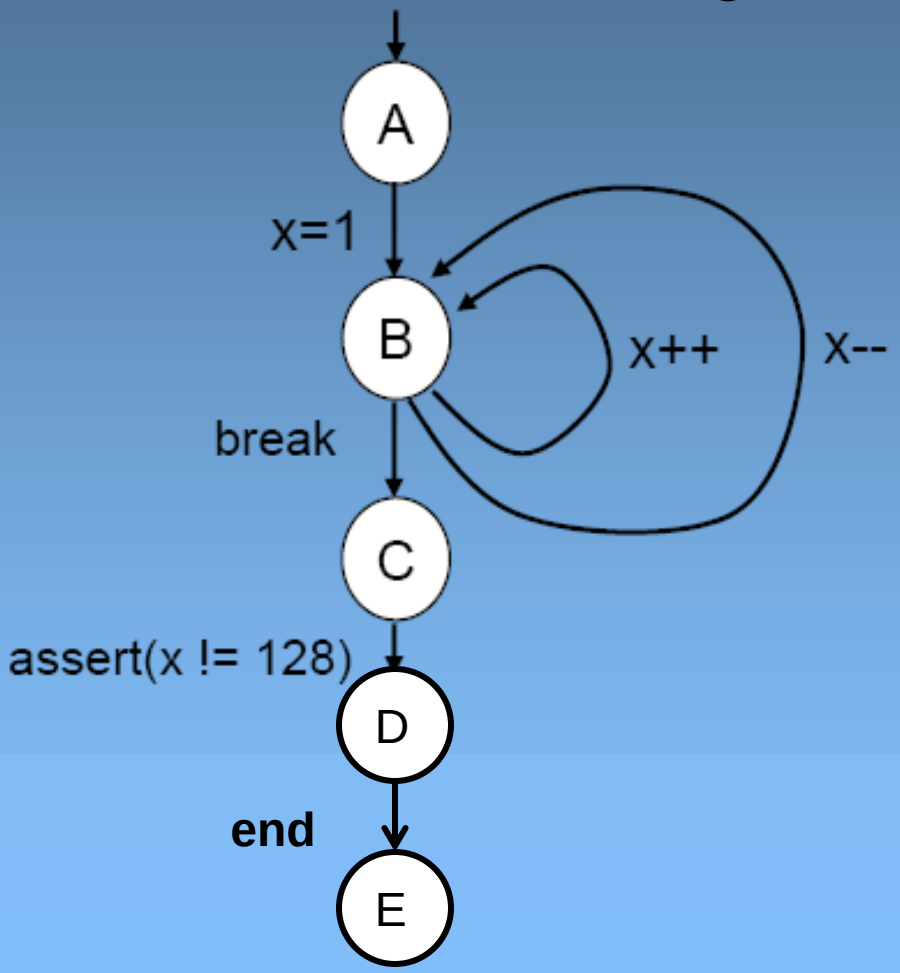
```
byte x;

A:   x = 1;
B:   do
       :: x++
       :: x--
       :: break
       od;
C:   assert(x != 128)
```

Q1: how many system states do you think this model defines?

5

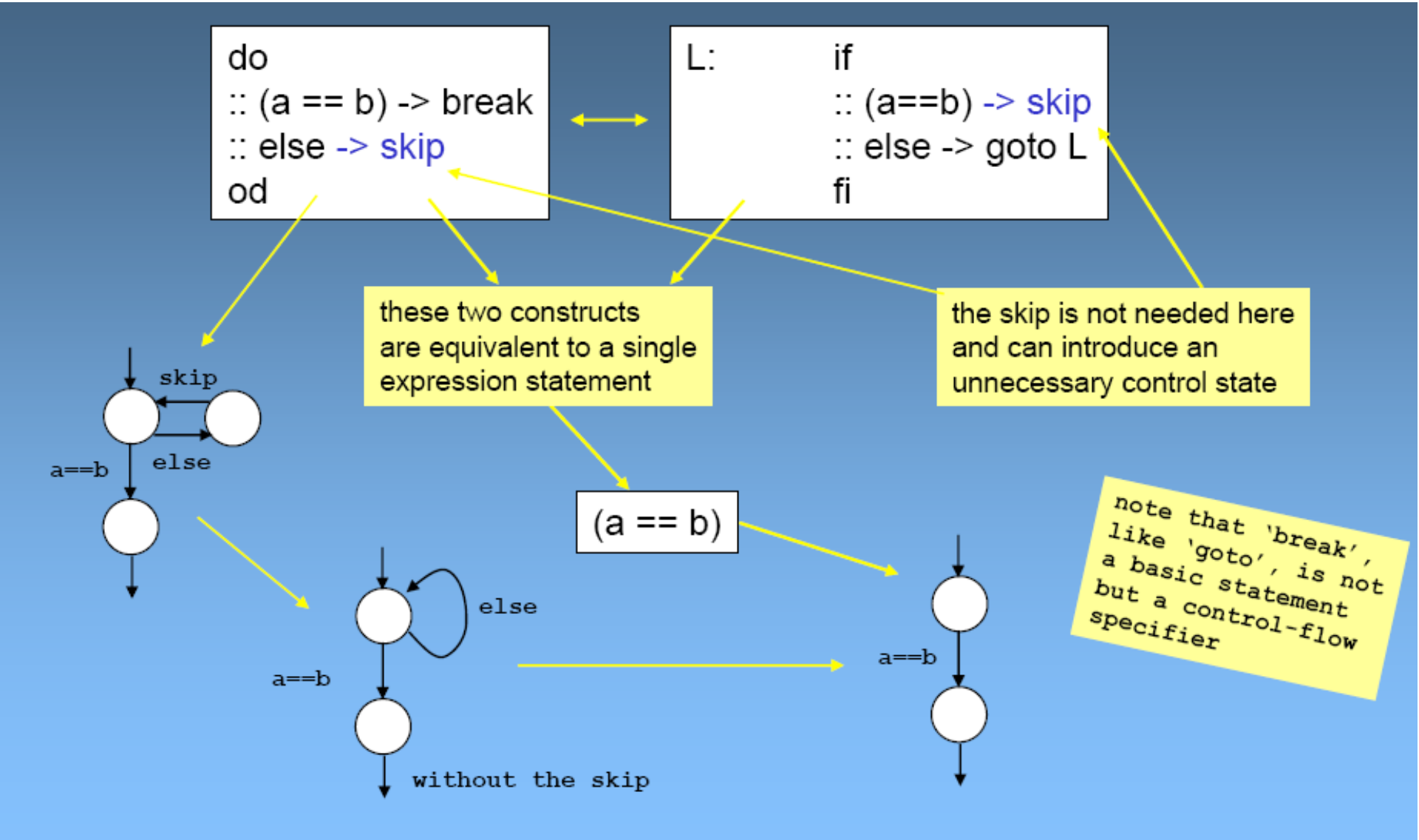
Q2: can the assertion be violated?

Q3: is the x-- statement needed?



Exploiting Executability Rules

– waiting for (a==b)



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;
}
```

Channels

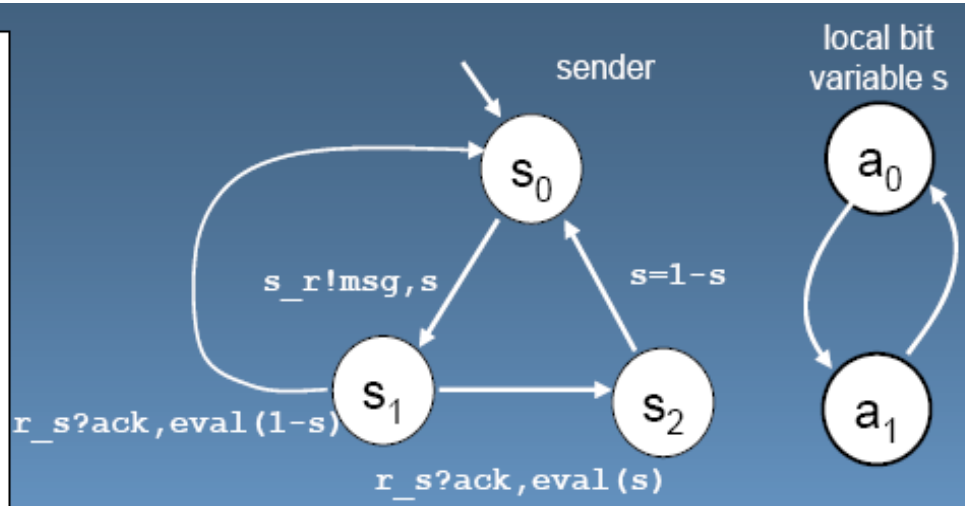
- Communication between processes is via channels, either for message passing or rendezvous (just set <dim> = 0 for a handshake).
- chan <name> = [<dim>] of { $<type_1>$, .., $<type_n>$ }
  - <name> = name of the channel
  - $<type_i>$ = type of elements to be transmitted
  - <dim> = maximum number of elements in the channel
- Example:
  - mtype = { DATA, ACK } ;
  - chan c = [5] of {mtype, bit};
  - sender executes: c ! DATA, 1;
  - receiver executes: c ? x, y; followed by: c ! ACK, y;

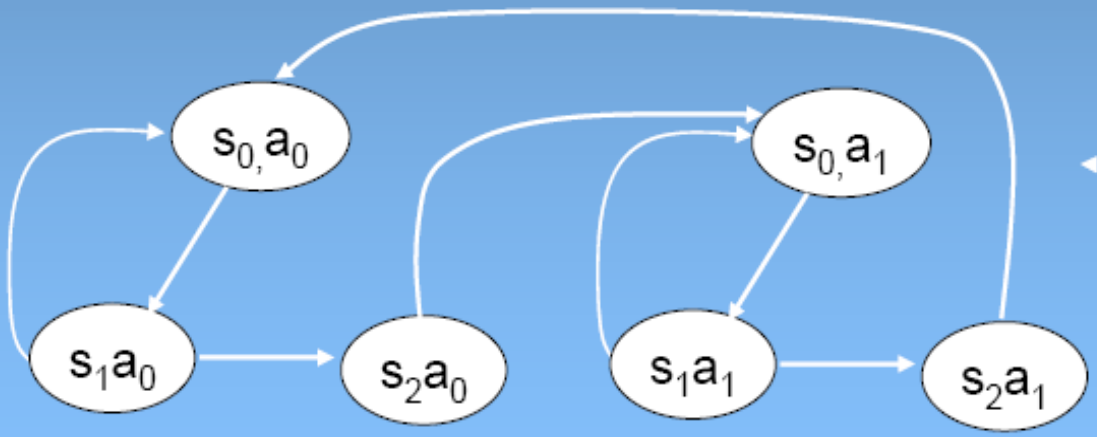
Example: Alternating Bit Protocol

```
mtype = { msg, ack };
chan s_r = [2] of { mtype, bit };
chan r_s = [2] of { mtype, bit };

active proctype sender()
{       bit seqno;
        do
        :: s_r!msg,seqno ->
                if
                :: r_s?ack,eval(seqno) ->
                        seqno = 1 - seqno /* fetch new msg */
                :: r_s?ack,eval(1-seqno)
                fi
        od
}

active proctype receiver()
{       bit expect, seqno;
        do
        :: s_r?msg,seqno ->
                r_s!ack,seqno;
                if
                :: seqno == expect      /* store msg */
                :: else                 /* ignore    */
                fi
        od
}
```

Automata View

```
active proctype sender()
{       bit s;
        do
        :: s_r!msg,s ->
                if
                :: r_s?ack,eval(s) ->
                        s = 1 - s
                :: r_s?ack,eval(1-s)
                fi
        od
}
```





sender and s together...
as a pure state automaton

eval( ) function

ch!msg(12)

ch?msg(eval (x))

maps the current value of x to a constant
to serve as a constraint on the receive statement

receive statement is executable
if the variable x equals 12

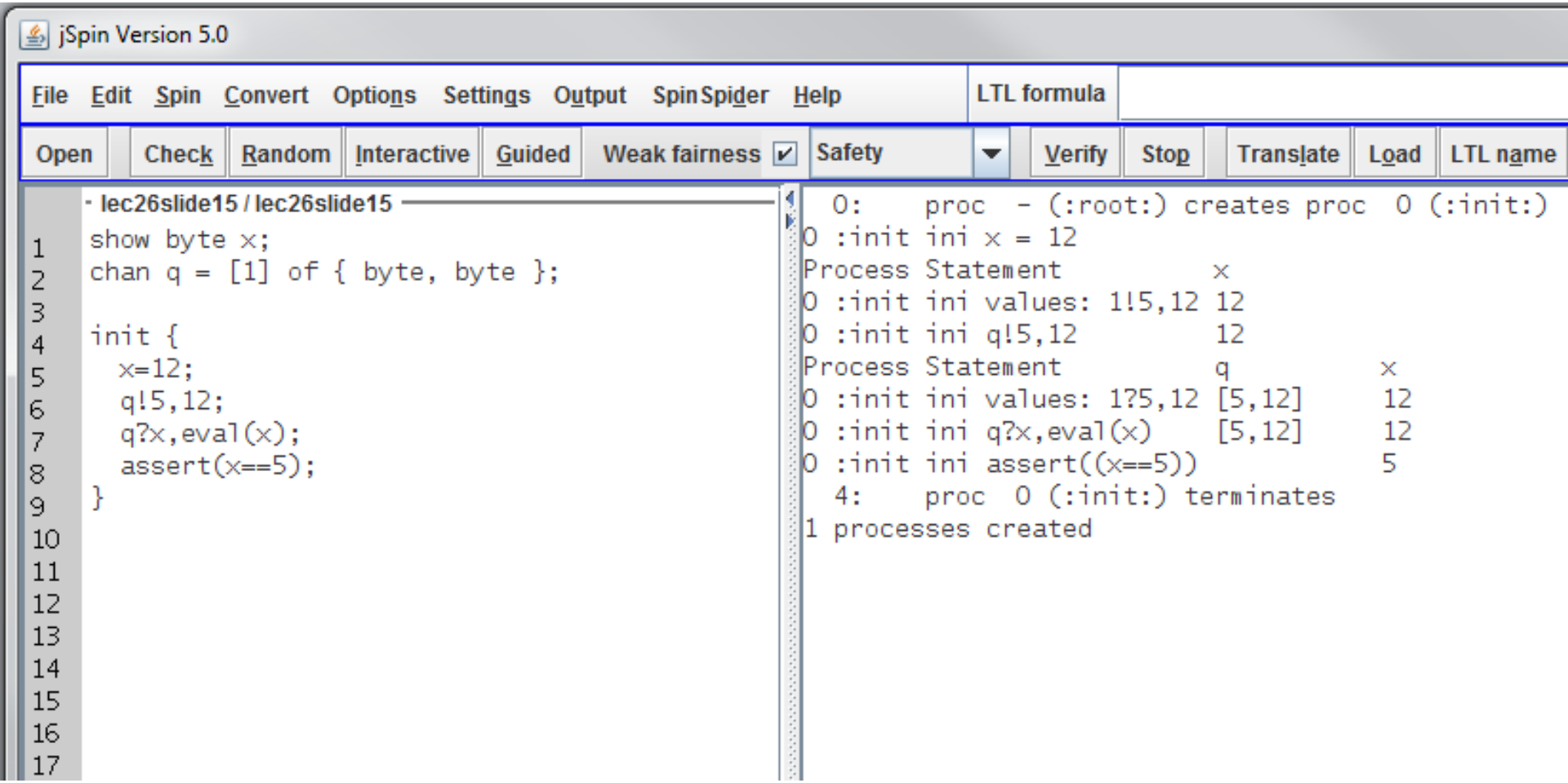
```
chan q = [1] of { byte, byte };

x = 12;
q!5(12);          /* same as writing: q!5,12 */
q?x(eval(x))      /* same as writing: q?x,eval(x) */
```

What is the value of x after the receive statement is executed?

Hint: evaluate right-to-left…

e statement

SPIN Simulation Output



Simulation Run

```
$ spin -u20 -c abp          # first 20 steps only
proc 0 = sender
proc 1 = receiver
q\p   0   1
  1   s_r!msg,0
  1   .   s_r?msg,0
  2   .   r_s!ack,0
  2   r_s?ack,0
  1   s_r!msg,1
  1   .   s_r?msg,1
  2   .   r_s!ack,1
  2   r_s?ack,1
-------------
depth-limit (-u20 steps) reached
-------------
final state:
-------------
#processes: 2
                queue 1 (s_r):
                queue 2 (r_s):
 20:    proc  1 (receiver) line  18 "abp" (state 7)
 20:    proc  0 (sender) line   6 "abp" (state 7)
2 processes created
```

Default Verification

```
$ spin -a abp.pml
$ gcc -o pan pan.c
$ ./pan
(Spin Version 4.1.0 -- 19 November 2003)
        + Partial Order Reduction

Full statespace search for:
        never claim             - (none specified)
        assertion violations    +
        acceptance   cycles     - (not selected)
        invalid end states      +

State-vector 60 byte, depth reached 11, errors: 0
      12 states, stored
       2 states, matched
      14 transitions (= stored+matched)
       0 atomic steps
hash conflicts: 0 (resolved)
(max size 2^18 states)

1.573   memory usage (Mbyte)

unreached in proctype sender
        line 11, state 5, "-end-"
        (1 of 5 states)
unreached in proctype receiver
        line 19, state 5, "-end-"
        (1 of 5 states)
```

algorithm used

properties checked

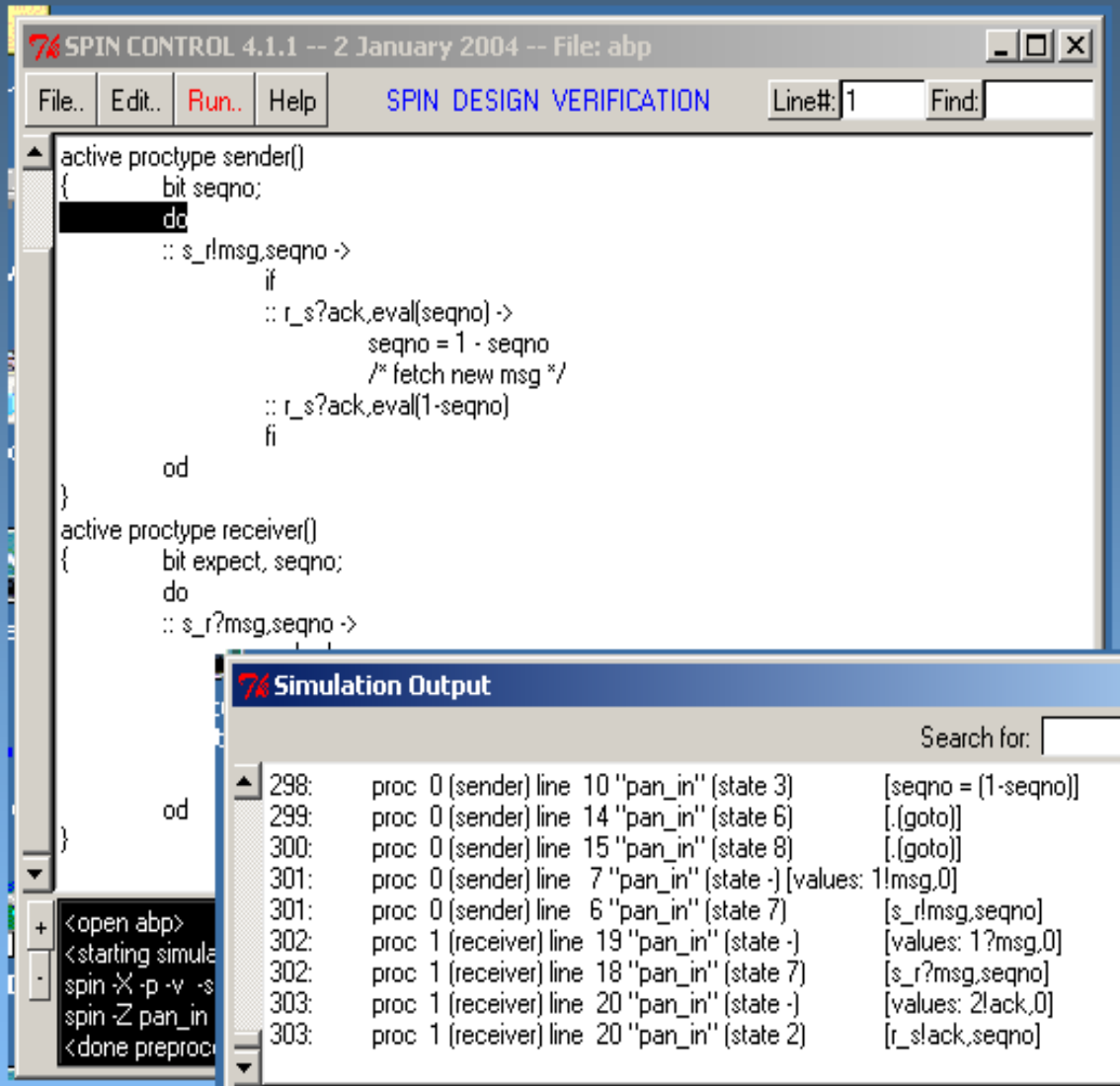
result: no errors…

amount of work done
(computation of a p.o. reduction of the global state space)

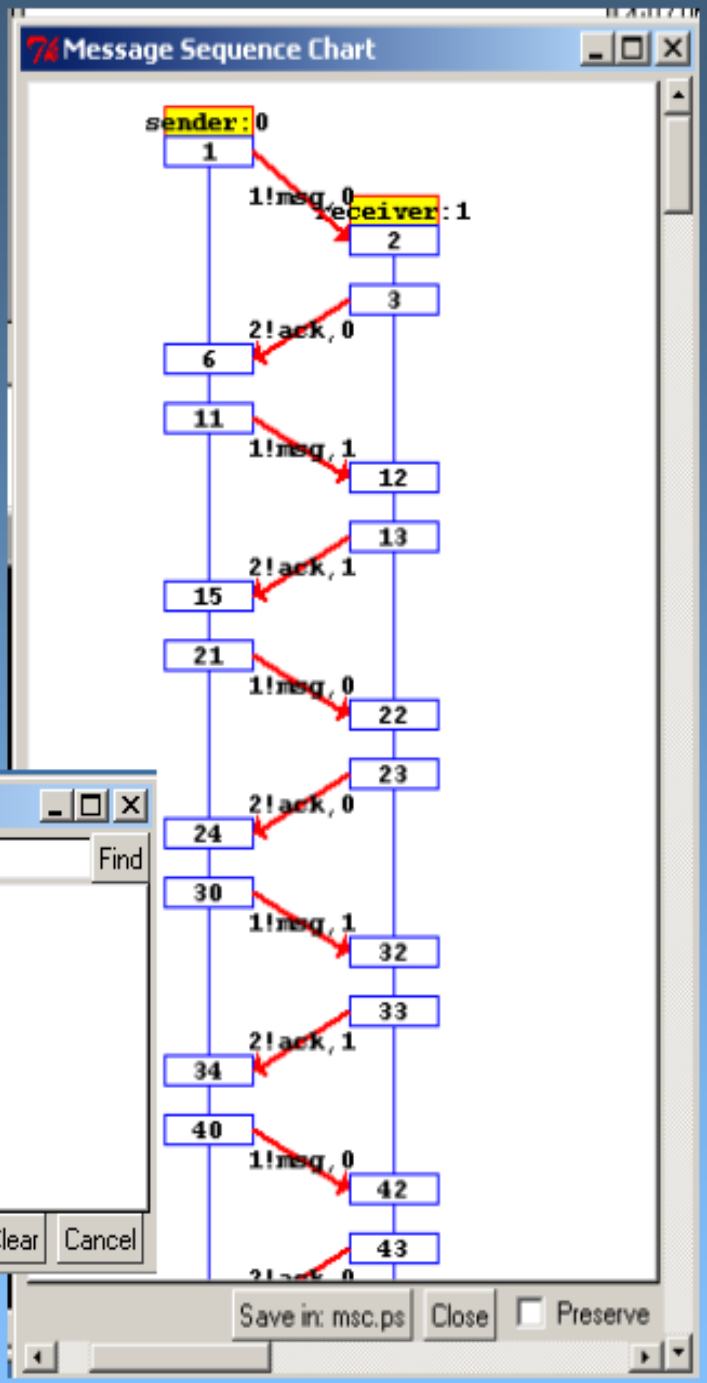
mem. resources used

unreachable code detected
(the processes do no terminate)

xspin


SPIN CONTROL 4.1.1 -- 2 January 2004 -- File: abp
File..
Edit..
Run..
Help
SPIN DESIGN VERIFICATION
Line#: 1
Find:
active proctype sender()
{ bit seqno;
do
:: s_r!msg,seqno ->
if
:: r_s?ack,eval(seqno) ->
seqno = 1 - seqno
/* fetch new msg */
:: r_s?ack,eval(1-seqno)
fi
od
}
active proctype receiver()
{ bit expect, seqno;
do
:: s_r?msg,seqno ->
od
}
<open abp>
<starting simula
spin -X -p -v -s
spin -Z pan_in
<done preproc

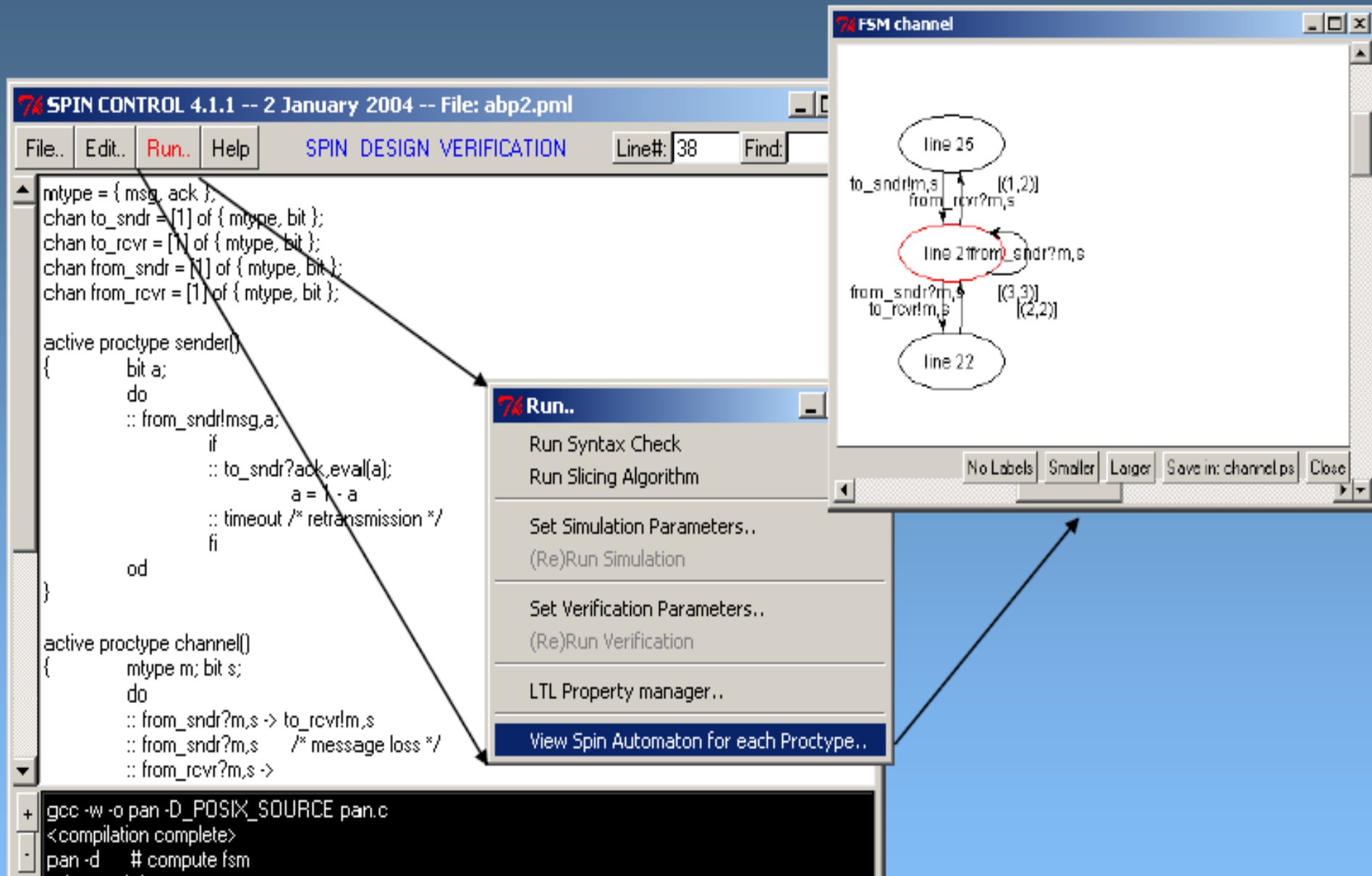
Simulation Output
Search for:
Find
298: proc 0 (sender) line 10 "pan_in" (state 3) [seqno = (1-seqno)]
299: proc 0 (sender) line 14 "pan_in" (state 6) [.(goto)]
300: proc 0 (sender) line 15 "pan_in" (state 8) [.(goto)]
301: proc 0 (sender) line 7 "pan_in" (state -) [values: 1!msg,0]
301: proc 0 (sender) line 6 "pan_in" (state 7) [s_r!msg,seqno]
302: proc 1 (receiver) line 19 "pan_in" (state -) [values: 1?msg,0]
302: proc 1 (receiver) line 18 "pan_in" (state 7) [s_r?msg,seqno]
303: proc 1 (receiver) line 20 "pan_in" (state -) [values: 2!ack,0]
303: proc 1 (receiver) line 20 "pan_in" (state 2) [r_s!ack,seqno]
Single Step
Run
Save in:
sim.out
Clear
Cancel


Message Sequence Chart
sender: 0
receiver: 1
1
1!msg,0
2
3
2!ack,0
6
11
1!msg,1
12
13
2!ack,1
15
21
1!msg,0
22
23
2!ack,0
24
30
1!msg,1
32
33
2!ack,1
34
40
1!msg,0
42
43
Save in: msc.ps
Close
Preserve

modelling message loss

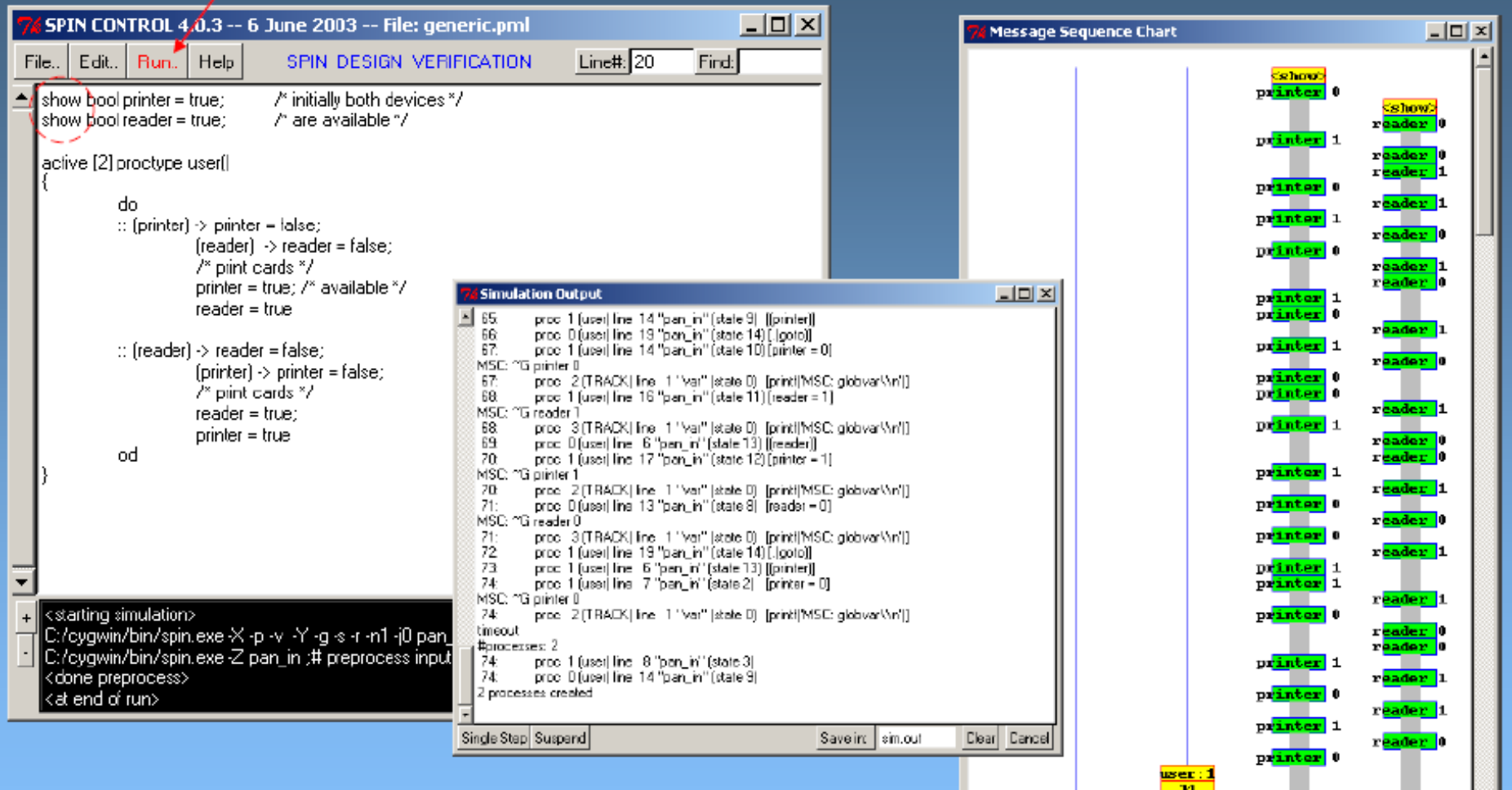
```
mtype = { msg, ack };
chan s_c = [2] of { mtype, bit };
chan c_r = [2] of { mtype, bit };
chan c_s = [2] of { mtype, bit };
chan r_c = [2] of { mtype, bit };

active proctype sender()
{  bit seqno;
   do
   :: s_c!msg,seqno ->
            if
            :: c_s?ack,eval(seqno) -> seqno = 1 - seqno /* new msg */
            :: c_s?ack,eval(1-seqno)
            fi
   od
}
active proctype channel()
{  mtype m; bit s;
   do
   :: s_c?m,s -> c_r!m,s       /* error-free transmission */
   :: s_c?m,s                  /* message loss */
   :: r_c?m,s -> c_s!m,s       /* error-free */
   od
}
active proctype receiver()
{  bit expect, seqno;
   do
   :: c_r?msg,seqno ->
            r_c!ack,seqno;
            if
            :: seqno == expect      /* store msg */
            :: else                 /* ignore    */
            fi
   od
}
```

viewing the automata with xspin



XSpin show statement

the spin gui



Promela Statements

- skip - always executable
- assert(expression) - always executable
- assignment statements - always executable
- if statement - executable if at least one guard is
- do statement - executable if at least one guard is
- break statement - always executable
- send (ch!) - executable if channel ch is not full
- receive (ch?) - executable if channel ch is not empty

Mutual Exclusion – test with assert( )
(Peterson's Solution Revisited)

```
bool turn, flag[2];
byte ncrit;

active [2] proctype user()
{
     assert(_pid == 0 || _pid == 1);
again:
     flag[_pid] = 1;
     turn = _pid;
     (flag[1 - _pid] == 0 || turn == 1 - _pid);
     ncrit++;
          assert(ncrit == 1);  /* critical section */
     ncrit--;
     flag[_pid] = 0;
     goto again
}
```

Atomic and Deterministic Step (d_step)

d_step sequences

more restrictive and more efficient than atomic sequences

```
d_step { guard -> stmnt1; stmnt2; ... stmntn }
```

- like an atomic, but *must be deterministic* and *may not block* anywhere inside the sequence
- especially useful to perform intermediate computations with a deterministic result, in a single indivisible step

```
d_step { /* reset array elements to 0 */
        i = 0;
        do
        :: i < N -> x[i] = 0; i++
        :: else -> break
        od;
        i = 0
      }
```

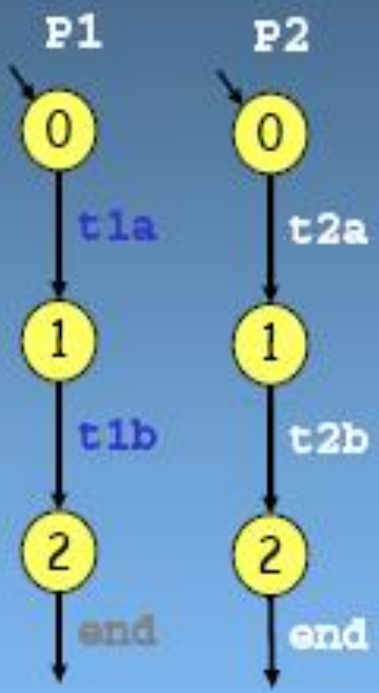
- atomic and d_step sequences are often used as a model reduction method, to lower complexity of large models (improving tractability)

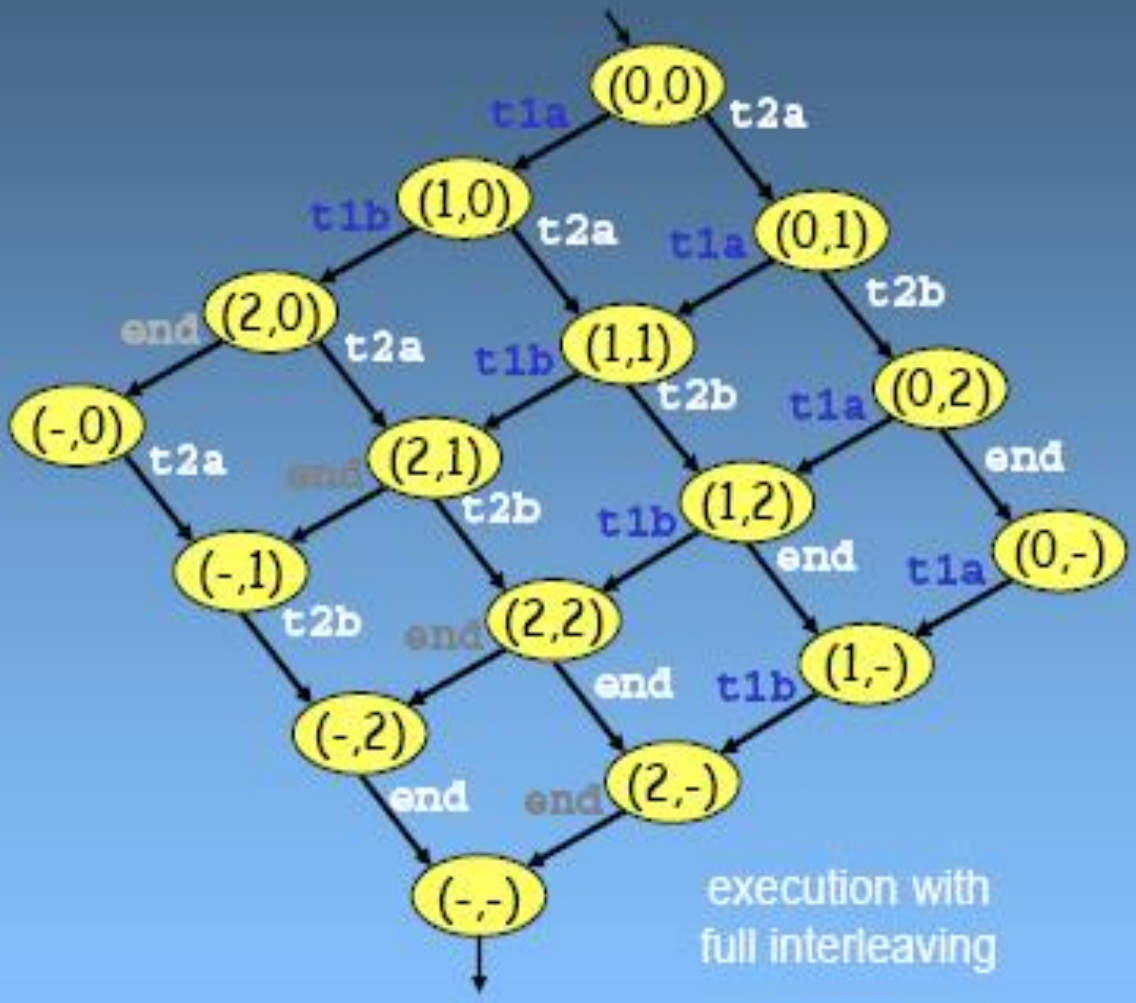
atomic and d_step

- both sequences are executable only when the *first* (guard) statement is executable
  - **atomic**: if any other statement blocks, atomicity is lost at that point; it can be regained once the statement becomes executable later
  - **d_step**: it is an *error* if any statement other than the guard statement blocks
- other differences:
  - **d_step**: the entire sequence is executed as *one* single transition
  - **atomic**: the sequence is executed step-by-step, but without interleaving; non-deterministic choices inside an atomic sequence are allowed
- caution:
  - infinite loops inside atomic or d_step sequences *are not* detected
  - the execution of this type of sequence models an indivisible step, which means that it cannot be infinite

```
active proctype P1() { t1a; t1b }
active proctype P2() { t2a; t2b }
```

execution
without atomics or d_steps



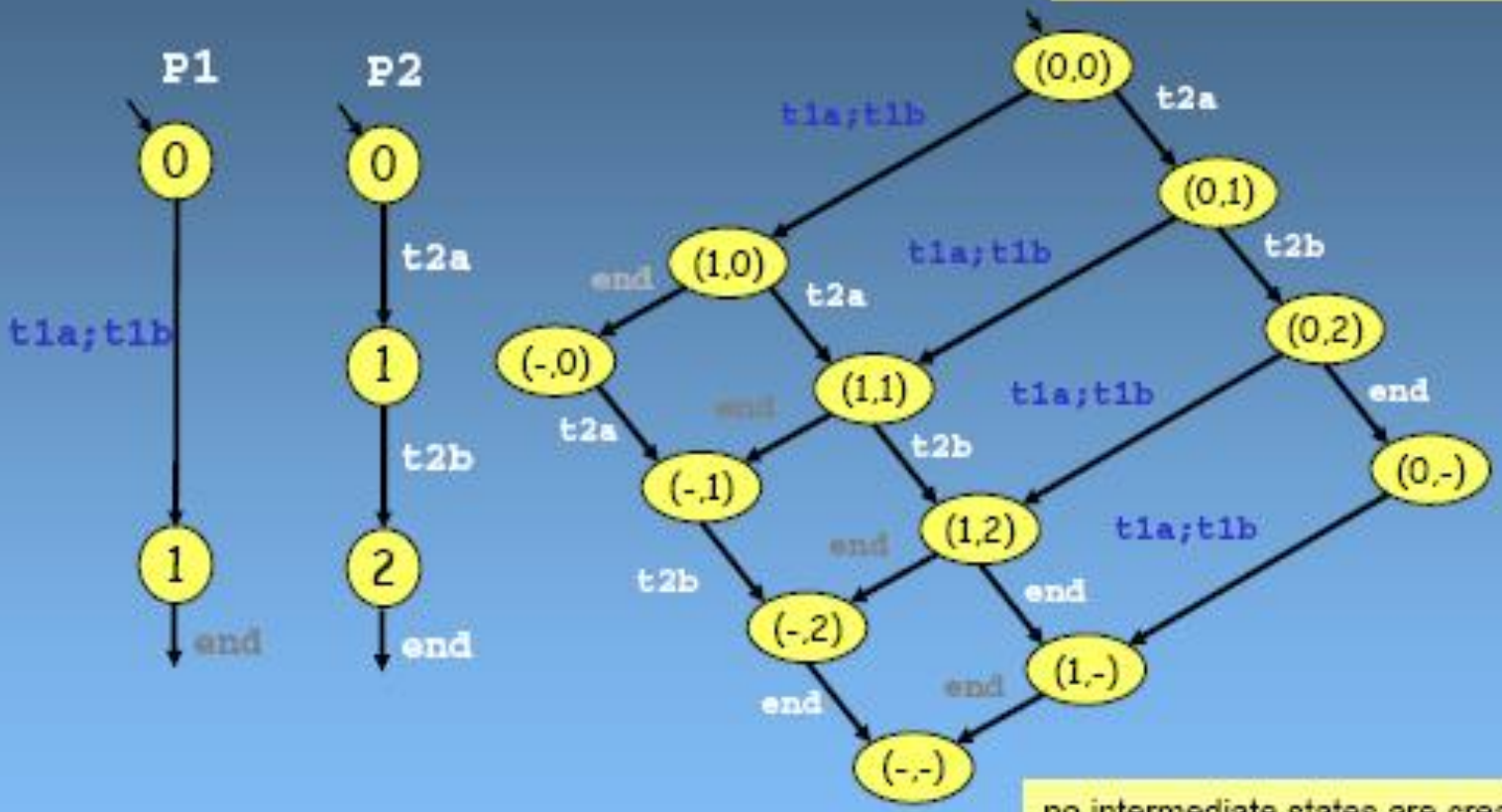


execution with
full interleaving

```
active proctype P1() { d_step {t1a; t1b} }
active proctype P2() { t2a; t2b }
```

execution with a d_step sequence

P1 now has only one transition…



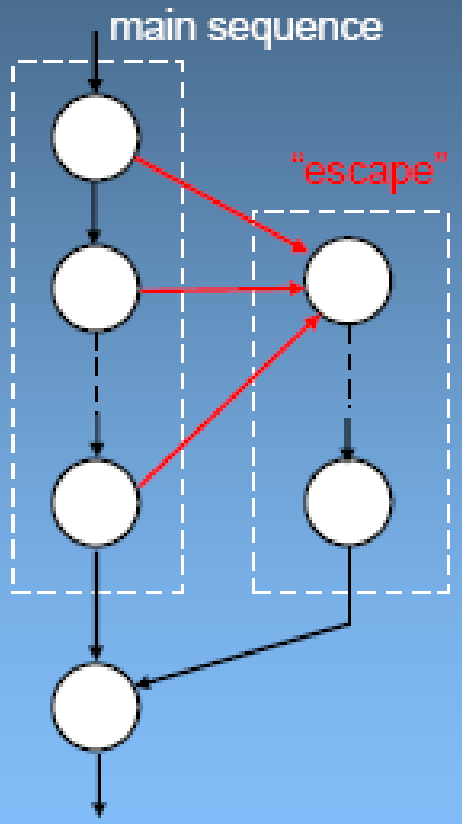
no intermediate states are created: faster, smaller graph, but no non-determinism possible inside d_step sequence itself

the last control construct: unless sequences

(cf. book, fig. 3.1, p. 63)

```
active proctype pots()
{               chan who;
idle:           line?offhook,who ->
                {               who!dialtone;
                                who?number;
                                if
                                :: who!busy
                                :: who!ringing;
                                                who!connected;
                                                who!hungup;
                                fi;
                                goto wait
                } unless {
                                if
                                :: who?hangup -> goto idle
                                :: timeout -> goto wait
                                fi
                }
wait:           who?hangup;
                goto idle
}
```

Higher priority block:



Model Checking Example

example: the Microsoft Zune 30 GB

zune

SOFTWARE MP3 PLAYERS DISCOVER SUPPORT
USER'S GUIDE TECHNICAL ISSUES FORUMS PRODUCT MANUALS

zune 30 faq

My Zune 30 is frozen. What should I do?

Follow these steps:

1. Disconnect your Zune from USB and AC power sources.
2. Because the player is frozen, its battery will drain—this is good. Wait until the battery is empty and the screen goes black. If the battery was fully charged, this might take a couple of hours.
3. Wait until after noon GMT on January 1, 2009 (that's 7 a.m. Eastern or 4 a.m. Pacific time).
4. Connect your Zune to either a USB port on the back or your computer or to AC power using the Zune AC Adapter and let it charge.

Once the battery has sufficient power, the player should start normally. No other action is required—you can go back to using your

. Wait until after noon GMT on January 1, 2009 (that's 7 a.m. Eastern or 4 a.m. Pacific tin

My Zune 30 has been working fine today. Should I be worried?

Nope, your Zune is fine and will continue to work as long as you do not connect it to your computer before noon GMT on January 1, 2009 (7 a.m. Eastern or 4 a.m. Pacific time).

Note: If you connect your player to a computer before noon GMT on January 1, 2009, you'll experience the freeze mentioned above—even if that computer does not have the Zune software installed. If this happens, follow the above steps.

What if I have rights-managed (DRM) content on my Zune?

Most likely, rights-managed content will not be affected by this issue. However, it's a good idea to sync your Zune with your computer once the freeze has been resolved, just to make sure your usage rights are up to date.

the code fragment

input : days elapsed since Jan 1, 1980
output: year + day of year

For Dec. 31, 2008, days = 10593

```
year = 1980;
while (days > 365)
{
    if (IsLeapYear(year))
    {
        if (days > 366)
        {
            days -= 366;
            year += 1;
        }
    } else
    {
        days -= 365;
        year += 1;
    }
}
```

Q: December 31, 2008 was the 366th day of the year.
(2008 was a leap year: a multiple of 4, but not of 100 or 400).

Q: How many test-cases would the developer have needed to test this code?

In general, though, how many test-cases would a developer need to fully test:

- a square-root function
- a sorting routine
- a mutual exclusion algorithm
- a lock-free fifo queue algorithm

```c
#include <stdio.h>

int IsLeapYear(int year)
{
  if ((year%4==0)&&((year%100!=0)||(year%400==0)))
    return 1;
  else
    return 0;
}

main()
{
  int year = 1980;
  int days = (2009-1980)*365 + (2008-1980)/4 + 1;

  printf("days = %d\n", days);

  while (days > 365)
  {
   if (IsLeapYear(year))
   {
     if (days > 366)
     {
        days -= 366;
        year += 1;
     }
   }
   else
   {
     days -= 365;
     year += 1;
   }
  }
  printf("year = %d, day of year = %d\n", year,days);
```

Infinite loop with days = 366, year = 2008…

zune resumed

```
 1 #define IsLeapYear(y) (((y%4 ==
 2
 3 chan q = [0] of { short };
 4
 5 active proctype zune()
 6 {    short year = 1980;
 7      short days;
 8
 9 end: do
10      :: q?days ->
11 S:      do
12         :: days > 365 ->
13            if
14            :: IsLeapYear(year)
15               if
16               :: days > 366 ->
17                  days = days - 366;
18                  year++
19               :: else /* do nothing */
20               fi
21            :: else ->
22               days = days - 365;
23               year++
24            fi
25         :: else ->
26            break
27         od;
28 E:      printf("Year: %d, Day %d\n", year, days)
29     od
30 }
```

```
init {
    /* jan 1, 2008 */
    short days = (2008-1980)*365 + (2008-1980)/4;
    if
    :: q!(days + 365)
    :: q!(days + 366)
    :: q!(days + 367)
    fi
}

#define at_S        zune@S
#define at_E        zune@E

#include "zune.ltl"
```

zune resumed

```
 1 #define IsLeapYear(y) (((y%4 == 0) && (y%100 != 0)) || (y%400 == 0)
 2
 3 chan q = [0] of { short };
 4
 5 active proctype zune()
 6 {    short year = 1980;
 7      short days;
 8
 9 end: do
10      :: q?days ->
11 S:       do
12          :: days > 365
13             if
14             :: IsLeapY
15                if
16                :: days
17                   days
18                   year
19                :: else
20                fi
21             :: else ->
22                days = days - 365;
23                year++
24             fi
25          :: else ->
26             break
27          od;
28 E:       printf("Year: %d, Day %d\n", year, days)
29     od
30 }
```

```
$ spin –f '![] (at_S  ->  <> at_E)' > zune.ltl
$ spin –a zune.pml      # create model checker
$ cc –o pan pan.c       # compile
$ ./pan –a              # run (0.031 sec)
$ spin –t –p –l zune.pml            # replay error trace
...
 <<<<<START OF CYCLE>>>>>
175:   proc  0 (zune) line  12 "zune.pml" (state 2)   [(days>365)]
177:   proc  0 (zune) line  14 "zune.pml" (state 3)   [IsLeapYear(year)]
179:   proc  0 (zune) line  21 "zune.pml" (state 7)   [else]
spin: trail ends after 179 steps
```

Frog Pond Puzzle

Consider the frog pond shown below. Three female frogs are on the three stones on the right and three male frogs are on the three stones on the left. Find a way to exchange the positions of the male and female frogs, so that the male frogs are all on the right and the females are all on the left. (You may first want to try it online at: http://www.hellam.net/maths2000/frogs.html)

The constraints that your solution must satisfy are as follows: frogs can only jump in the direction they are facing. They can either jump one rock forward if the next rock is empty or they can jump over a frog if the next rock has a frog on it and the rock after it is empty; they can jump over one frog.

jSpin Version 5.0

File Edit Spin Convert Options Settings Output SpinSpider Help | LTL formula []!success

Open | Check | Random | Interactive | Guided | Weak fairness ☑ | Safety | Verify | Stop | Translate

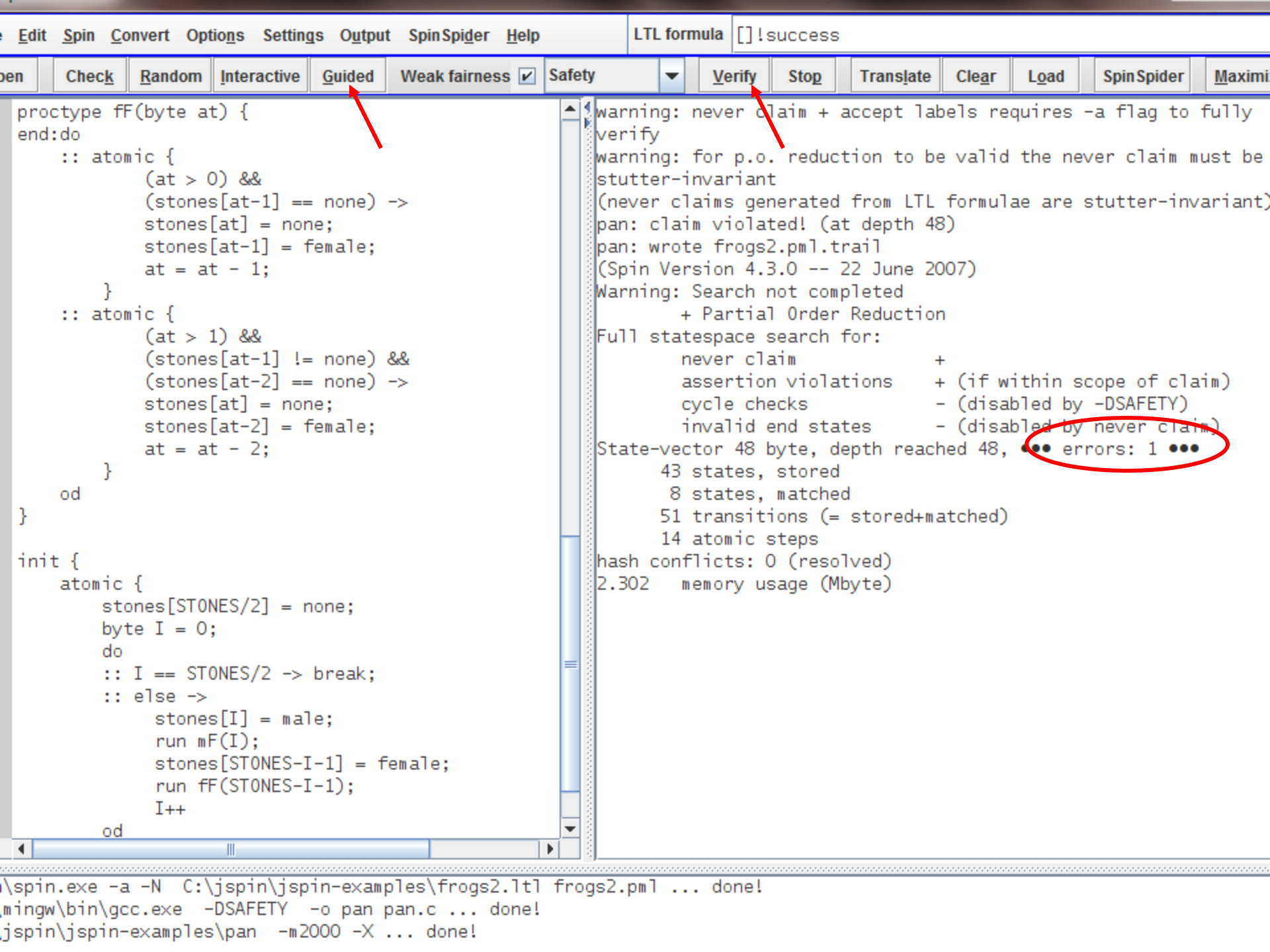
frogs.pml / -

```
1  #define STONES 7
2  #define success (\
3      (stones[0]==female) && \
4      (stones[1]==female) && \
5      (stones[2]==female) && \
6      (stones[4]==male) && \
7      (stones[5]==male) && \
8      (stones[6]==male) )
9
10 mtype = { none, male, female };
11
12 mtype stones[STONES];
13
14 proctype mF(byte at) {
15 end:do
16     :: atomic {
17         (at < STONES-1) &&
18         (stones[at+1] == none) ->
19         stones[at] = none;
20         stones[at+1] = male;
21         at = at+1;
22        }
23     :: atomic {
24         (at < STONES-2) &&
25         (stones[at+1] != none) &&
26         (stones[at+2] == none) ->
27         stones[at] = none;
28         stones[at+2] = male;
29         at = at+2;
30        }
31     od
32 }
```

```
spin: couldn't find claim 3 (ignore
0 :init ini stones[[(7/2)]
Process Statement          stones[0
0 :init ini else           0
0 :init ini stones[I] =[I] 0
Starting mF with pid 2
0 :init ini run mF(I)      male
0 :init ini sto[((7-I)-1)] male
Starting fF with pid 3
0 :init ini run fF(((7-I)- male
0 :init ini I = (I+1)      male
Process Statement          :init:(0
0 :init ini else           1
0 :init ini stones[I] =[I] 1
Starting mF with pid 4
0 :init ini run mF(I)      1
0 :init ini sto[((7-I)-1)] 1
Starting fF with pid 5
0 :init ini run fF(((7-I)- 1
0 :init ini I = (I+1)      1
0 :init ini else           2
0 :init ini stones[I] =[I] 2
Starting mF with pid 6
0 :init ini run mF(I)      2
0 :init ini sto[((7-I)-1)] 2
Starting fF with pid 7
0 :init ini run fF(((7-I)- 2
0 :init ini I = (I+1)      2
0 :init ini I==(7/2)       3
0 :init ini break          3
6 fF    38  at>0)&[(at-1)] 3
6 fF    39  stones[at][at] 3
6 fF    40  stones[(at-1)] 3
6 fF    41  at = (at-1)    3
```

```
}

proctype fF(byte at) {
end:do
    :: atomic {
        (at > 0) &&
        (stones[at-1] == none) ->
        stones[at] = none;
        stones[at-1] = female;
        at = at-1;
       }
    :: atomic {
        (at > 1) &&
        (stones[at-1] != none) &&
        (stones[at-2] == none) ->
        stones[at] = none;
        stones[at-2] = female;
        at = at-2;
       }
    od
}

init {
    byte I=0;
    atomic {
        stones[STONES/2]=none;
        do
        :: I == STONES/2 -> break;
        :: else ->
            stones[I] = male;
            run mF(I);
            stones[STONES-I-1] = female;
            run fF(STONES-I-1);
            I++
        od
    }
}
```

```
Process Statement          stones[0]  stones
0 :init ini else           0          0
0 :init ini stones[I] =[I] 0          0
Starting mF with pid 2
0 :init ini run mF(I)      male       0
0 :init ini sto[((7-I)-1)] male       0
Starting fF with pid 3
0 :init ini run fF(((7-I)- male       0
0 :init ini I = (I+1)      male       0
Process Statement          :init:(0): stones
0 :init ini else           1          male
0 :init ini stones[I] =[I] 1          male
Starting mF with pid 4
0 :init ini run mF(I)      1          male
0 :init ini sto[((7-I)-1)] 1          male
Starting fF with pid 5
0 :init ini run fF(((7-I)- 1          male
0 :init ini I = (I+1)      1          male
0 :init ini else           2          male
0 :init ini stones[I] =[I] 2          male
Starting mF with pid 6
0 :init ini run mF(I)      2          male
0 :init ini sto[((7-I)-1)] 2          male
Starting fF with pid 7
0 :init ini run fF(((7-I)- 2          male
0 :init ini I = (I+1)      2          male
0 :init ini I==(7/2)       3          male
0 :init ini break          3          male
6 fF    38  at>0)&[(at-1)] 3          male
6 fF    39  stones[at][at] 3          male
6 fF    40  stones[(at-1)] 3          male
6 fF    41  at = (at-1)    3          male
Process Statement          :init:(0): fF(6)
stones[6]
5 mF    26  (at<(7-2))&&(s 3          3
female
5 mF    27  stones[at][at] 3          3
female
```


Edit Spin Convert Options Settings Output SpinSpider Help
LTL formula []!success
Check Random Interactive Guided Weak fairness Safety Verify Stop Translate Clear Load SpinSpider
proctype fF(byte at) {
end:do
:: atomic {
(at > 0) &&
(stones[at-1] == none) ->
stones[at] = none;
stones[at-1] = female;
at = at - 1;
}
:: atomic {
(at > 1) &&
(stones[at-1] != none) &&
(stones[at-2] == none) ->
stones[at] = none;
stones[at-2] = female;
at = at - 2;
}
od
}
init {
atomic {
stones[STONES/2] = none;
byte I = 0;
do
:: I == STONES/2 -> break;
:: else ->
stones[I] = male;
run mF(I);
stones[STONES-I-1] = female;
run fF(STONES-I-1);
I++
od
warning: never claim + accept labels requires -a flag to fully
verify
warning: for p.o. reduction to be valid the never claim must be
stutter-invariant
(never claims generated from LTL formulae are stutter-invariant)
pan: claim violated! (at depth 48)
pan: wrote frogs2.pml.trail
(Spin Version 4.3.0 -- 22 June 2007)
Warning: Search not completed
+ Partial Order Reduction
Full statespace search for:
never claim +
assertion violations + (if within scope of claim)
cycle checks - (disabled by -DSAFETY)
invalid end states - (disabled by never claim)
State-vector 48 byte, depth reached 48, ••• errors: 1 •••
43 states, stored
8 states, matched
51 transitions (= stored+matched)
14 atomic steps
hash conflicts: 0 (resolved)
2.302 memory usage (Mbyte)
\spin.exe -a -N C:\jspin\jspin-examples\frogs2.ltl frogs2.pml ... done!
mingw\bin\gcc.exe -DSAFETY -o pan pan.c ... done!
jspin\jspin-examples\pan -m2000 -X ... done!

```
    male      male      male      none      female    female    female
    male      male      male      none      female    female    female
    male      male      male      none      female    female    female
    male      male      male      none      female    female    female
    male      male      male      none      female    female    female
    male      male      male      none      female    female    female
    male      male      male      none      none      female    female
    male      male      male      female    none      female    female
0): fF(6):at  stones[0]  stones[1]  stones[2]  stones[3]  stones[4]  stones[5]  stones[6]
    3         male      male      male      female    none      female    female
    3         male      male      male      female    none      female    female
    3         male      male      none      female    none      female    female
    3         male      male      none      female    male      female    female
0): fF(6):at  mF(5):at  stones[0]  stones[1]  stones[2]  stones[3]  stones[4]  stones[5]  stones[6]
    3         4         male      male      none      female    male      female    female
    3         4         male      male      none      female    male      female    female
    3         4         male      none      none      female    male      female    female
    3         4         male      none      male      female    male      female    female
0): fF(6):at  mF(3):at  mF(5):at  stones[0]  stones[1]  stones[2]  stones[3]  stones[4]  stones[5]  stones[6]
    3         2         4         male      none      male      female    male      female    female
    3         2         4         male      none      male      female    male      female    female
    3         2         4         male      none      male      none      male      female    female
    3         2         4         male      female    male      none      male      female    female
    1         2         4         male      female    male      none      male      female    female
    1         2         4         male      female    male      none      male      female    female
    1         2         4         male      female    male      none      male      none      female
    1         2         4         male      female    male      female    male      none      female
0): fF(4):at  fF(6):at  mF(3):at  mF(5):at  stones[0]  stones[1]  stones[2]  stones[3]  stones[4]  stones[5]  stones[6]
    3         1         2         4         male      female    male      female    male      none      female
    3         1         2         4         male      female    male      female    male      none      female
    3         1         2         4         male      female    male      female    male      none      none
    3         1         2         4         male      female    male      female    male      female    none
0): fF(2):at  fF(4):at  fF(6):at  mF(3):at  mF(5):at  stones[0]  stones[1]  stones[2]  stones[3]  stones[4]  stones[5]  stones
    5         3         1         2         4         male      female    male      female    male      female     none
    5         3         1         2         4         male      female    male      female    male      female     none
    5         3         1         2         4         male      female    male      female    none      female     none
    5         3         1         2         4         male      female    male      female    none      female     male
    5         3         1         2         6         male      female    male      female    none      female     male
    5         3         1         2         6         male      female    male      female    none      female     male
```

2	4	male	female	male	female	none	female	none	
2	4	male	female	male	female	none	female	male	
2	6	male	female	male	female	none	female	male	
2	6	male	female	male	female	none	female	male	
2	6	male	female	none	female	none	female	male	
2	6	male	female	none	female	male	female	male	
4	6	male	female	none	female	male	female	male	
4	6	male	female	none	female	male	female	male	
4	6	none	female	none	female	male	female	male	
4	6	none	female	male	female	male	female	male	

mF(1):at	mF(3):at	mF(5):at	stones[0]	stones[1]	stones[2]	stones[3]	stones[4]	stones[5]	stones[6]
2	4	6	none	female	male	female	male	female	male
2	4	6	none	female	male	female	male	female	male
2	4	6	none	none	male	female	male	female	male
2	4	6	female	none	male	female	male	female	male
2	4	6	female	none	male	female	male	female	male
2	4	6	female	none	male	female	male	female	male
2	4	6	female	none	male	none	male	female	male
2	4	6	female	female	male	none	male	female	male
2	4	6	female	female	male	none	male	female	male
2	4	6	female	female	male	none	male	female	male
2	4	6	female	female	male	none	male	none	male
2	4	6	female	female	male	female	male	none	male
2	4	6	female	female	male	female	male	none	male
2	4	6	female	female	male	female	male	none	male
2	4	6	female	female	male	female	none	none	male
2	4	6	female	female	male	female	none	male	male
2	5	6	female	female	male	female	none	male	male
2	5	6	female	female	male	female	none	male	male
2	5	6	female	female	none	female	none	male	male
2	5	6	female	female	none	female	male	male	male

mF(1):at	mF(3):at	mF(5):at	stones[0]	stones[1]	stones[2]	stones[3]	stones[4]	stones[5]	stones[6]
4	5	6	female	female	none	female	male	male	male
4	5	6	female	female	none	female	male	male	male
4	5	6	female	female	none	none	male	male	male
4	5	6	female	female	female	none	male	male	male

Frog Pond Puzzle

Frog Pond Puzzle

Frog Pond Puzzle

Frog Pond Puzzle

Frog Pond Puzzle

Frog Pond Puzzle

Frog Pond Puzzle

Frog Pond Puzzle

Frog Pond Puzzle

Frog Pond Puzzle

Frog Pond Puzzle

Frog Pond Puzzle

Frog Pond Puzzle

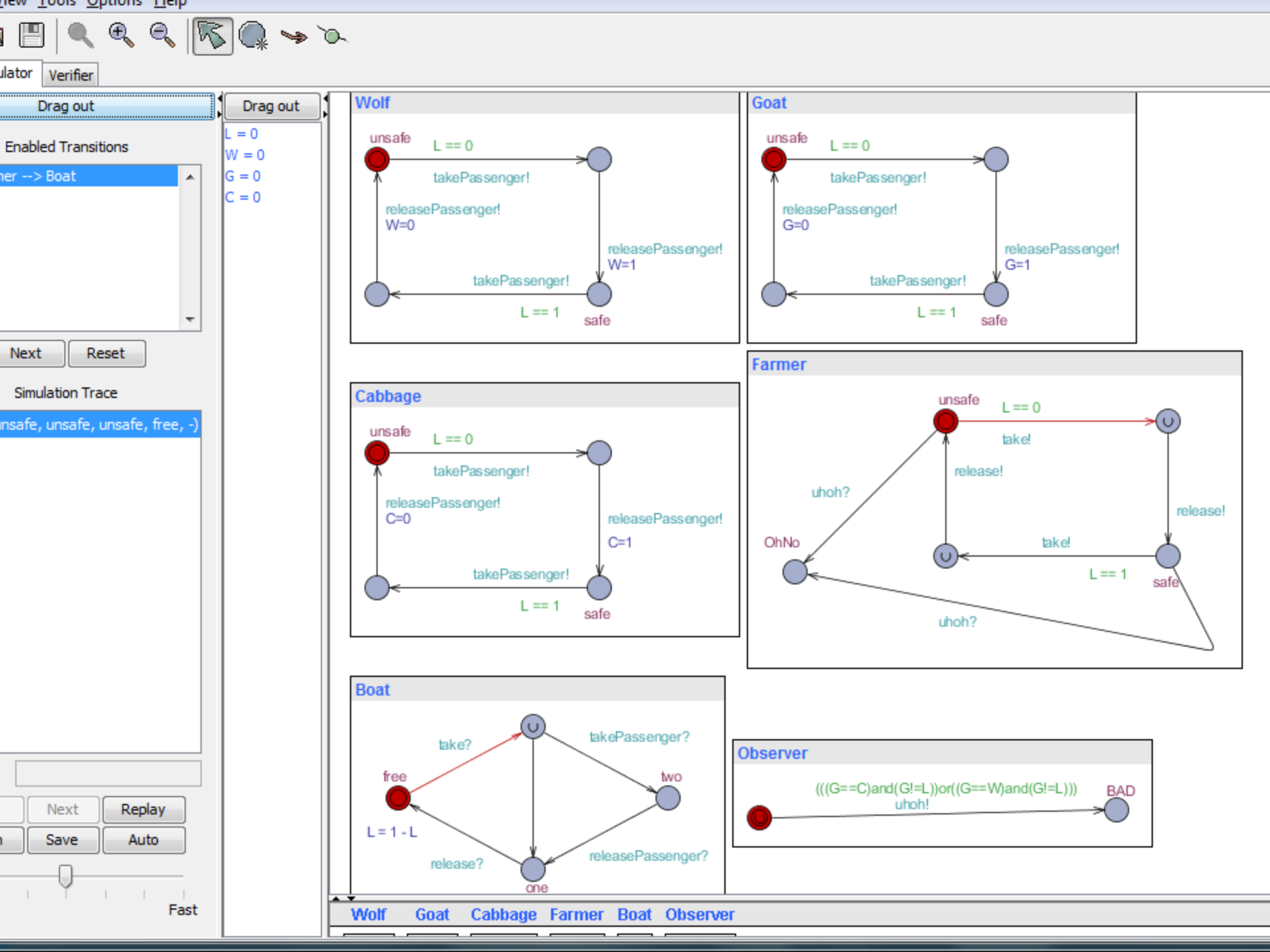
Frog Pond Puzzle

Frog Pond Puzzle

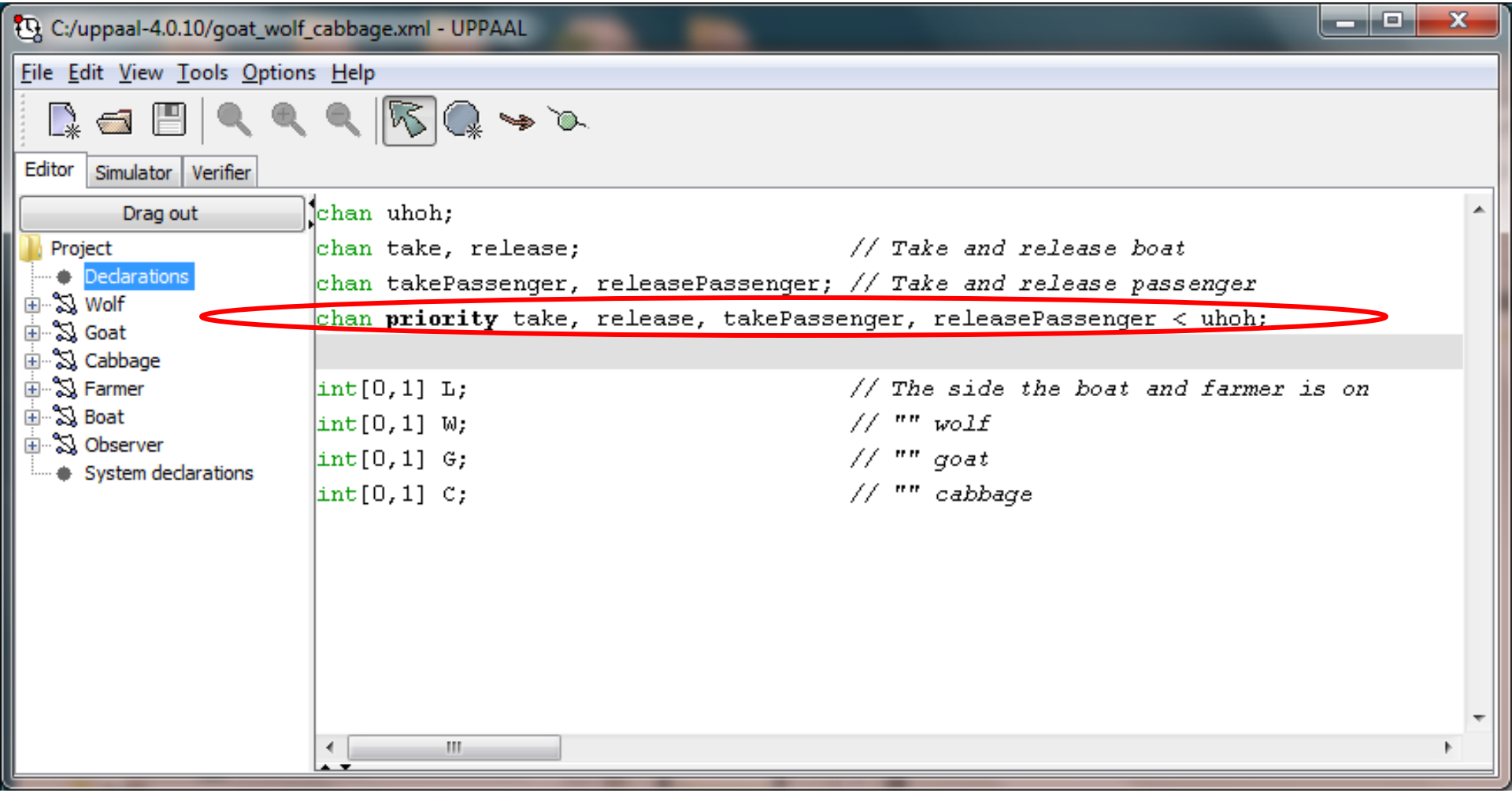
Frog Pond Puzzle

UPPAAL Example

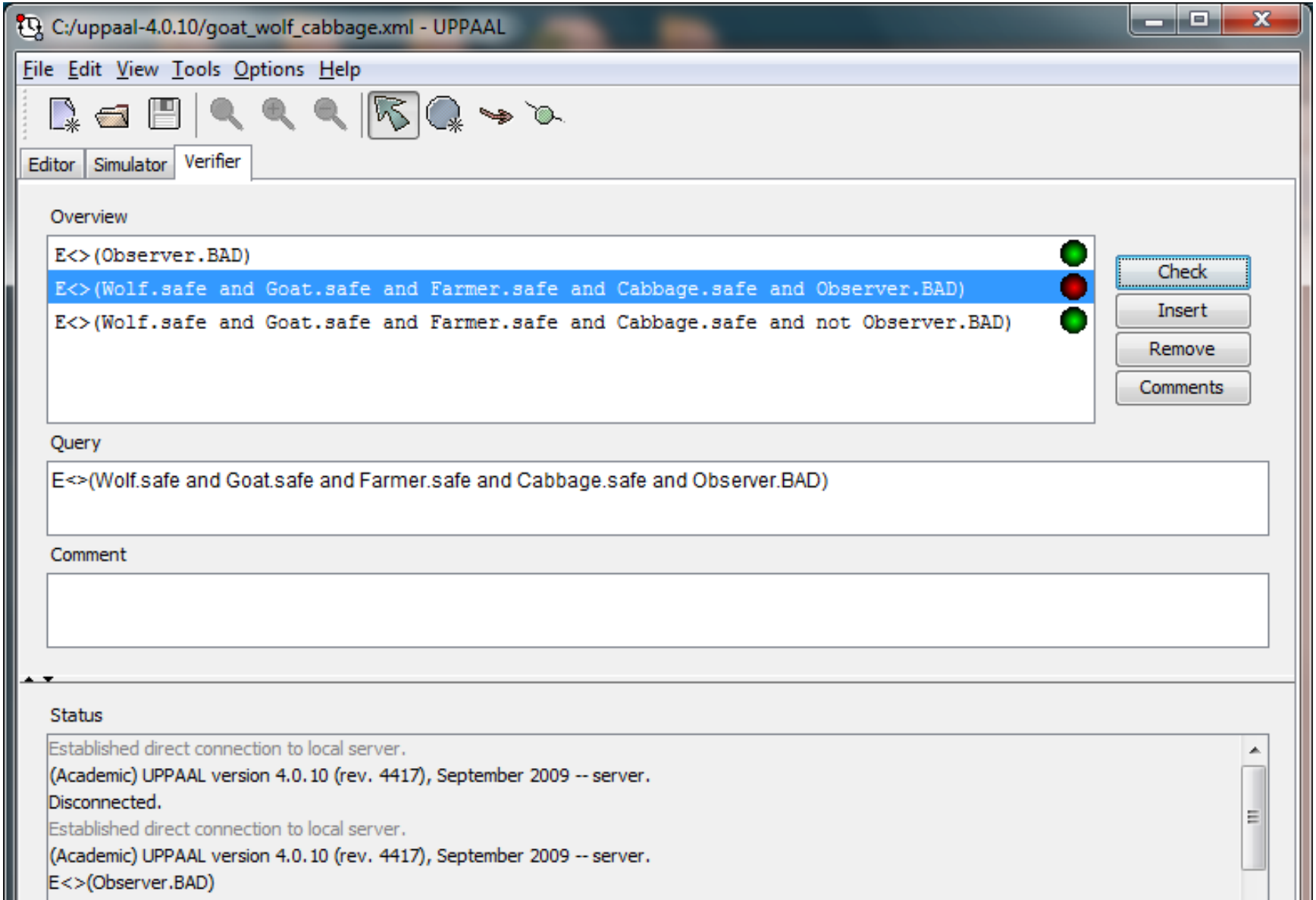
- Wolf, goat, cabbage, farmer problem
  - The farmer needs to move the wolf, goat, and cabbage from one side of the river to the other side. The farmer can only carry one passenger.
  - If the wolf and goat are left alone, the wolf will eat the goat. If the goat and cabbage are left alone, the goat will eat the cabbage.
  - How can the farmer transport the passengers without allowing one to be eaten?


Verifier
Drag out
Enabled Transitions
Boat
Next
Reset
Simulation Trace
Next
Replay
Save
Auto
Fast
Drag out
L = 0
W = 0
G = 0
C = 0
Wolf
unsafe
L == 0
takePassenger!
releasePassenger!
W=0
releasePassenger!
W=1
takePassenger!
L == 1
safe
Goat
unsafe
L == 0
takePassenger!
releasePassenger!
G=0
releasePassenger!
G=1
takePassenger!
L == 1
safe
Cabbage
unsafe
L == 0
takePassenger!
releasePassenger!
C=0
releasePassenger!
C=1
takePassenger!
L == 1
safe
Farmer
unsafe
L == 0
take!
release!
uhoh?
release!
OhNo
take!
L == 1
safe
uhoh?
Boat
take?
takePassenger?
free
two
L = 1 - L
release?
releasePassenger?
one
Observer
(((G==C)and(G!=L))or((G==W)and(G!=L)))
uhoh!
BAD
Wolf
Goat
Cabbage
Farmer
Boat
Observer

UPPAAL Model



UPPAAL Model



jSpin Version 4.7

File Edit Spin Convert Options Settings Output SpinSpider Help | LTL formula []oneLeft

Open | Check | Random | Interactive | Guided | Weak fairness ☑ | Safety | Verify | Stop | Translate | Clear | Load | Sp

- farmer.pml * / -

```
1  /* farmer.pml - solution to goat, wolf, cabbage, farmer problem */
2
3  /* possible positions for an object */
4  #define LEFT    0
5  #define RIGHT   1
6  #define oneLeft ((farmer==LEFT)||(goat==LEFT)||(wolf==LEFT)||(cabbage==LEFT))
7  /* initial positions of the wolf, goat, cabbage, and farmer */
8  byte wolf   = LEFT;
9  byte goat   = LEFT;
10 byte cabbage = LEFT;
11 byte farmer = LEFT;
12
13 bool bad = false;
14
15 #define print_state printf("farmer: %d wolf: %d goat: %d cabbage: %d bad: %d\n", farmer, wolf, goat, cabbage, bad)
16
17 #define move_right(var) {       \
18     d_step {            \
19         var = RIGHT;        \
20         farmer = RIGHT;     \
21         print_state     \
22     }               \
23 }
24
25 #define move_left(var) {        \
26     d_step {            \
27         var = LEFT;     \
28         farmer = LEFT;      \
29         print_state     \
30     }               \
31 }
32
33
```

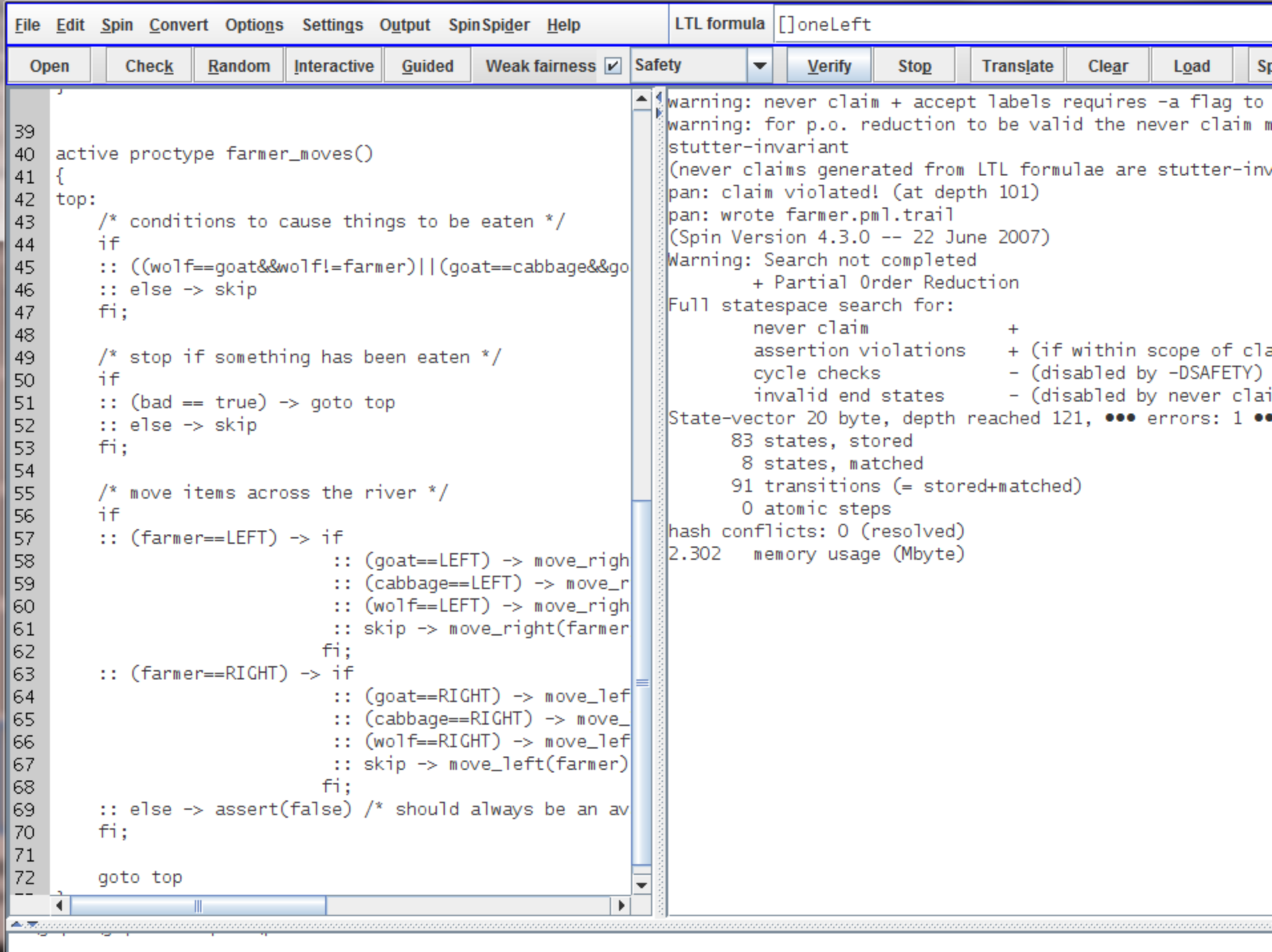
Saved farmer.ltl
bin\spin.exe -a -N C:\jspin\jspin-examples\farmer.ltl farmer.pml ... done!

File Edit Spin Convert Options Settings Output SpinSpider Help

LTL formula []oneLeft

Open | Check | Random | Interactive | Guided | Weak fairness ☑ | Safety | Verify | Stop | Translate | Clear | Load | Sp

```
39
40  active proctype farmer_moves()
41  {
42  top:
43      /* conditions to cause things to be eaten */
44      if
45      :: ((wolf==goat&&wolf!=farmer)||(goat==cabbage&&goat!=farmer))-> bad=true;
46      :: else -> skip
47      fi;
48
49      /* stop if something has been eaten */
50      if
51      :: (bad == true) -> goto top
52      :: else -> skip
53      fi;
54
55      /* move items across the river */
56      if
57      :: (farmer==LEFT) -> if
58                           :: (goat==LEFT) -> move_right(goat);
59                           :: (cabbage==LEFT) -> move_right(cabbage);
60                           :: (wolf==LEFT) -> move_right(wolf);
61                           :: skip -> move_right(farmer);
62                          fi;
63      :: (farmer==RIGHT) -> if
64                           :: (goat==RIGHT) -> move_left(goat);
65                           :: (cabbage==RIGHT) -> move_left(cabbage);
66                           :: (wolf==RIGHT) -> move_left(wolf);
67                           :: skip -> move_left(farmer);
68                          fi;
69      :: else -> assert(false) /* should always be an available move */
70      fi;
71
72      goto top
73  }
```


File Edit Spin Convert Options Settings Output SpinSpider Help
LTL formula []oneLeft
Open Check Random Interactive Guided Weak fairness
Safety Verify Stop Translate Clear Load
39
40 active proctype farmer_moves()
41 {
42 top:
43 /* conditions to cause things to be eaten */
44 if
45 :: ((wolf==goat&&wolf!=farmer)||(goat==cabbage&&go
46 :: else -> skip
47 fi;
48
49 /* stop if something has been eaten */
50 if
51 :: (bad == true) -> goto top
52 :: else -> skip
53 fi;
54
55 /* move items across the river */
56 if
57 :: (farmer==LEFT) -> if
58 :: (goat==LEFT) -> move_righ
59 :: (cabbage==LEFT) -> move_r
60 :: (wolf==LEFT) -> move_righ
61 :: skip -> move_right(farmer
62 fi;
63 :: (farmer==RIGHT) -> if
64 :: (goat==RIGHT) -> move_lef
65 :: (cabbage==RIGHT) -> move_
66 :: (wolf==RIGHT) -> move_lef
67 :: skip -> move_left(farmer)
68 fi;
69 :: else -> assert(false) /* should always be an av
70 fi;
71
72 goto top
warning: never claim + accept labels requires -a flag to
warning: for p.o. reduction to be valid the never claim m
stutter-invariant
(never claims generated from LTL formulae are stutter-inv
pan: claim violated! (at depth 101)
pan: wrote farmer.pml.trail
(Spin Version 4.3.0 -- 22 June 2007)
Warning: Search not completed
+ Partial Order Reduction
Full statespace search for:
never claim +
assertion violations + (if within scope of cla
cycle checks - (disabled by -DSAFETY)
invalid end states - (disabled by never clai
State-vector 20 byte, depth reached 121, ●●● errors: 1 ●●
83 states, stored
8 states, matched
91 transitions (= stored+matched)
0 atomic steps
hash conflicts: 0 (resolved)
2.302 memory usage (Mbyte)

File Edit Spin Convert Options Settings Output SpinSpider Help

LTL formula []oneLeft

Open | Check | Random | Interactive | Guided | Weak fairness ☑ | Safety | Verify | Stop | Translate | Clear | Load | Sp

```
0 farme 46  1
0 farme 51  farmer==0
0 farme 52  goat==0
0 farme 52  goat = 1
Process Statement          goat
0 farme 52  farmer = 1     1
farmer: 1 wolf: 0 goat: 1 cabbage: 0 bad: 0
Process Statement          farmer     goat
0 farme 52  printf('farmer 1          1
0 farme 40  else           1          1
0 farme 40  1              1          1
0 farme 46  else           1          1
0 farme 46  1              1          1
0 farme 57  farmer==1      1          1
0 farme 61  1              1          1
0 farme 61  farmer = 0     1          1
0 farme 61  farmer = 0     0          1
farmer: 0 wolf: 0 goat: 1 cabbage: 0 bad: 0
0 farme 61  printf('farmer 0          1
0 farme 40  else           0          1
0 farme 40  1              0          1
0 farme 46  else           0          1
0 farme 46  1              0          1
0 farme 51  farmer==0      0          1
0 farme 53  cabbage==0     0          1
0 farme 53  cabbage = 1    0          1
Process Statement          cabbage    farmer     goat
0 farme 53  farmer = 1     1          0          1
farmer: 1 wolf: 0 goat: 1 cabbage: 1 bad: 0
0 farme 53  printf('farmer 1          1          1
0 farme 40  else           1          1          1
0 farme 40  1              1          1          1
0 farme 46  else           1          1          1
0 farme 46  1              1          1          1
```

```
c:\mingw\bin\gcc.exe  -DSAFETY  -o pan pan.c ... done!
C:\jspin\jspin-examples\pan  -m2000 -X ... done!
bin\spin.exe -g -l -p -r -s -t -X -u250 farmer.pml ... done!
```

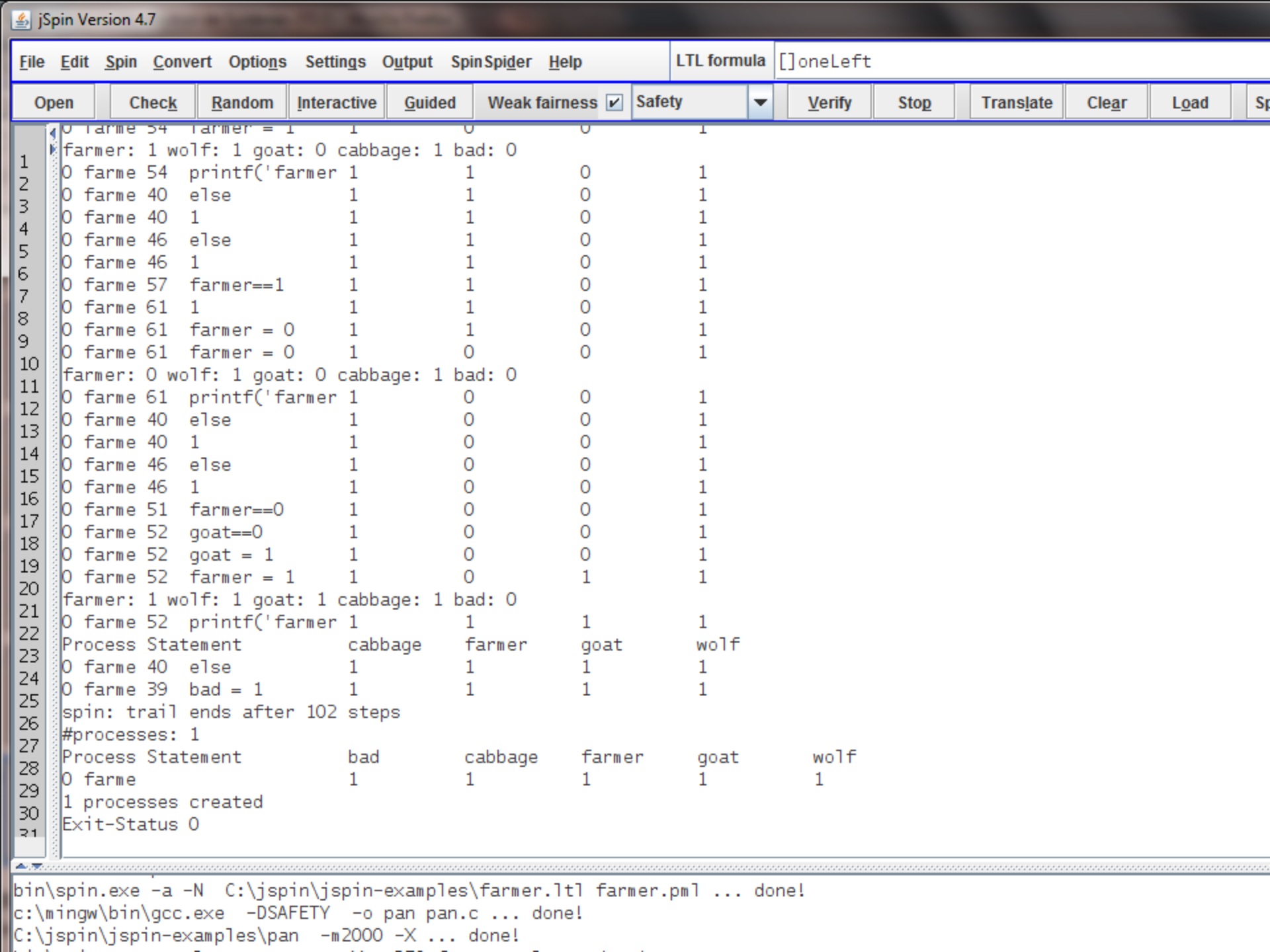
jSpin Version 4.7

File Edit Spin Convert Options Settings Output Spin Spider Help | LTL formula []oneLeft

Open | Check | Random | Interactive | Guided | Weak fairness ☑ | Safety | Verify | Stop | Translate | Clear | Load

```
farmer: 1 wolf: 0 goat: 1 cabbage: 1 bad: 0
1  0 farme 53  printf('farmer 1          1          1
2  0 farme 40  else           1          1          1
3  0 farme 40  1              1          1          1
4  0 farme 46  else           1          1          1
5  0 farme 46  1              1          1          1
6  0 farme 57  farmer==1      1          1          1
7  0 farme 58  goat==1        1          1          1
8  0 farme 58  goat = 0       1          1          1
9  0 farme 58  farmer = 0     1          1          0
10 farmer: 0 wolf: 0 goat: 0 cabbage: 1 bad: 0
11 0 farme 58  printf('farmer 1          0          0
12 0 farme 40  else           1          0          0
13 0 farme 40  1              1          0          0
14 0 farme 46  else           1          0          0
15 0 farme 46  1              1          0          0
16 0 farme 51  farmer==0      1          0          0
17 0 farme 54  wolf==0        1          0          0
18 0 farme 54  wolf = 1       1          0          0
19 Process Statement          cabbage    farmer     goat       wolf
20 0 farme 54  farmer = 1     1          0          0          1
21 farmer: 1 wolf: 1 goat: 0 cabbage: 1 bad: 0
22 0 farme 54  printf('farmer 1          1          0          1
23 0 farme 40  else           1          1          0          1
24 0 farme 40  1              1          1          0          1
25 0 farme 46  else           1          1          0          1
26 0 farme 46  1              1          1          0          1
27 0 farme 57  farmer==1      1          1          0          1
28 0 farme 61  1              1          1          0          1
29 0 farme 61  farmer = 0     1          1          0          1
30 0 farme 61  farmer = 0     1          0          0          1
31 farmer: 0 wolf: 1 goat: 0 cabbage: 1 bad: 0
```

```
bin\spin.exe -a -N  C:\jspin\jspin-examples\farmer.ltl farmer.pml ... done!
c:\mingw\bin\gcc.exe  -DSAFETY  -o pan pan.c ... done!
C:\jspin\jspin-examples\pan  -m2000 -X ... done!
```


jSpin Version 4.7
File Edit Spin Convert Options Settings Output SpinSpider Help
LTL formula
[]oneLeft
Open Check Random Interactive Guided Weak fairness
Safety
Verify Stop Translate Clear Load
0 farme 54 farmer = 1 1 0 0 1
farmer: 1 wolf: 1 goat: 0 cabbage: 1 bad: 0
0 farme 54 printf('farmer 1 1 0 1
0 farme 40 else 1 1 0 1
0 farme 40 1 1 1 0 1
0 farme 46 else 1 1 0 1
0 farme 46 1 1 1 0 1
0 farme 57 farmer==1 1 1 0 1
0 farme 61 1 1 1 0 1
0 farme 61 farmer = 0 1 1 0 1
0 farme 61 farmer = 0 1 0 0 1
farmer: 0 wolf: 1 goat: 0 cabbage: 1 bad: 0
0 farme 61 printf('farmer 1 0 0 1
0 farme 40 else 1 0 0 1
0 farme 40 1 1 0 0 1
0 farme 46 else 1 0 0 1
0 farme 46 1 1 0 0 1
0 farme 51 farmer==0 1 0 0 1
0 farme 52 goat==0 1 0 0 1
0 farme 52 goat = 1 1 0 0 1
0 farme 52 farmer = 1 1 0 1 1
farmer: 1 wolf: 1 goat: 1 cabbage: 1 bad: 0
0 farme 52 printf('farmer 1 1 1 1
Process Statement cabbage farmer goat wolf
0 farme 40 else 1 1 1 1
0 farme 39 bad = 1 1 1 1 1
spin: trail ends after 102 steps
#processes: 1
Process Statement bad cabbage farmer goat wolf
0 farme 1 1 1 1 1
1 processes created
Exit-Status 0
bin\spin.exe -a -N C:\jspin\jspin-examples\farmer.ltl farmer.pml ... done!
c:\mingw\bin\gcc.exe -DSAFETY -o pan pan.c ... done!
C:\jspin\jspin-examples\pan -m2000 -X ... done!