CS 6110 Software Correctness, Spring 2022

Ganesh Gopalakrishnan
School of Computing
University of Utah
Salt Lake City, UT 84112

URL: bit.ly/cs6110s22

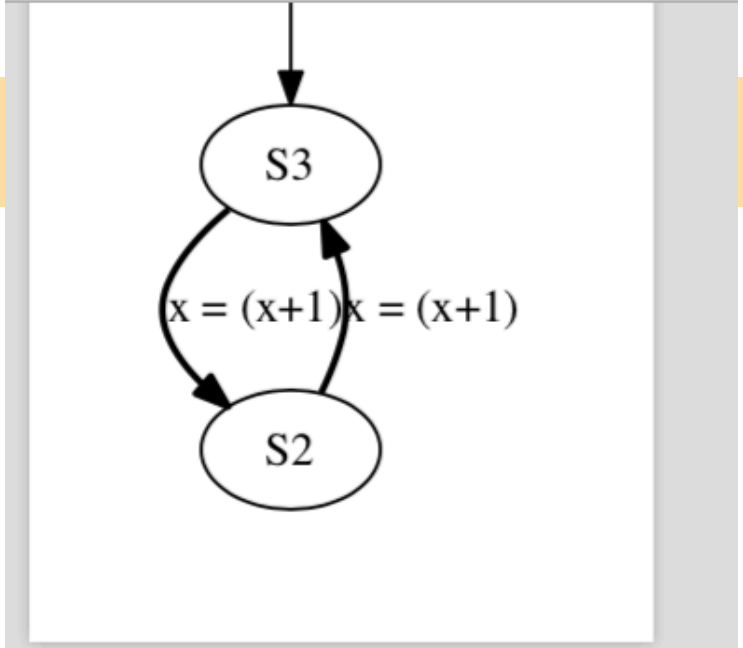
ex1

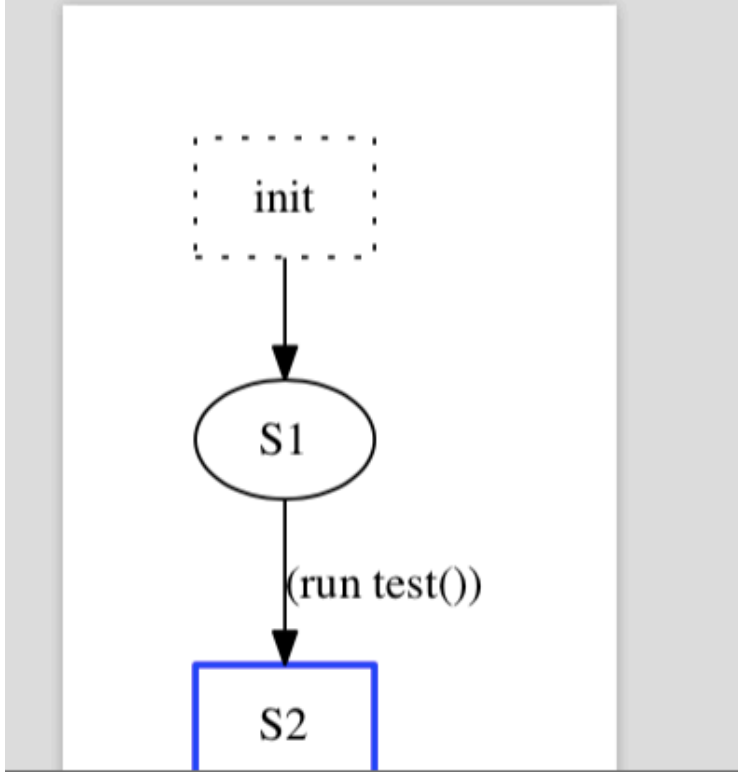
```
byte x; /* initialized to 0 by default */
proctype test()
{do
 :: x++ ; x++
 od
}
init {
 run test();
}
```

ex1

```
te=bold,shape=box];

exs]$ pan -D | dot -Tpdf > ex1.pdf
exs]$
  Bot L117755   [[(Shell:run)]] 5:33AM
```





ex1

```
warning: no accept labels are defined, so option -a has no effect (ignored)
(Spin Version 6.4.5 -- 1 January 2016)
        + Partial Order Reduction
Full statespace search for:
        never claim                   - (none specified)
        assertion violations        +
        acceptance   cycles           - (not selected)
        invalid end states        +
State-vector 20 byte, depth reached 2, ••• errors: 0 •••
        3 states, stored
        1 states, matched
        4 transitions (= stored+matched)
        0 atomic steps
hash conflicts:         0 (resolved)
Stats on memory usage (in Megabytes):
    0.000        equivalent memory usage for states (stored*(State-vector + overhead))
    0.292        actual memory usage for states
  128.000        memory used for hash table (-w24)
    1.068        memory used for DFS stack (-m20000)
  129.264        total actual memory usage
unreached in proctype test
        ex1.pml:5, state 6, "-end-"
        (1 of 6 states)
unreached in init
        (0 of 2 states)
pan: elapsed time 0 seconds
```

ex2 : fire up w/o init process

```
byte x;
active proctype test()
{do
 :: x++ ; x++
 od
}
```

ex3

```
byte x;
active proctype test()
{do
:: x++ ; x++
od }
never { do
:: skip
:: x==3 -> break
od
}
```

```
starting claim 1
using statement merging
Never claim moves to line 7        [(1)]
0 test: 1)  x = (x+1)
Process Statement          x
0 test: 1)  x = (x+1)      1
0 test: 1)  x = (x+1)      2
Never claim moves to line 8        [((x==3))]
0 test: 1)  x = (x+1)      3
spin: trail ends after 8 steps
#processes: 1
  8:        proc  0 (test:1) ex3.pml:3 (state 3)
MSC: ~G line 10
  8:        proc  - (never_0:1) ex3.pml:10 (state 7)
1 processes created
Exit-Status 0
```

ex4

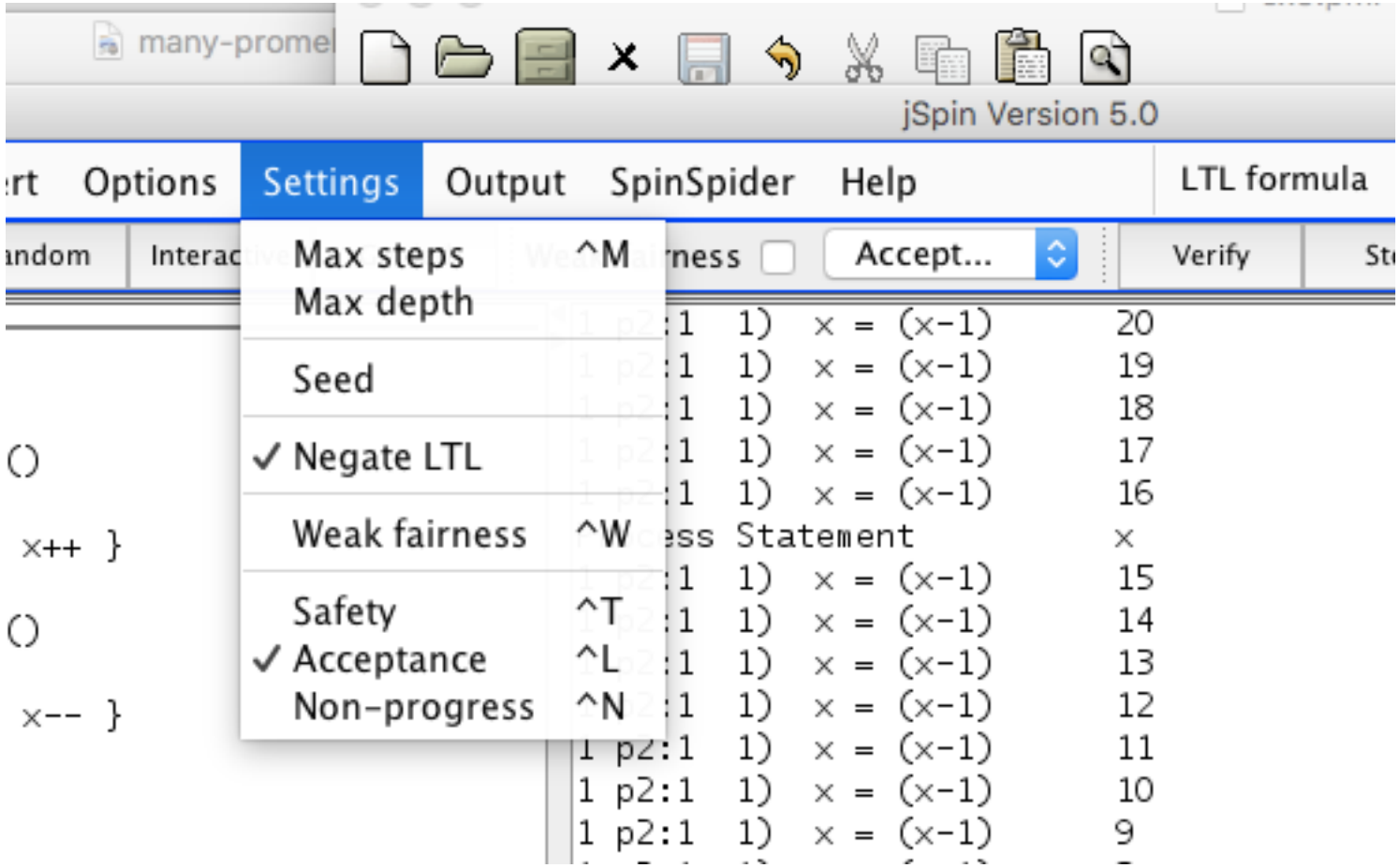
```
byte x;
active proctype test()
{do
 :: atomic { x++ ; x++ }
 od
}
never {
do
:: skip
:: x==3 -> break
od
}
```

ex5

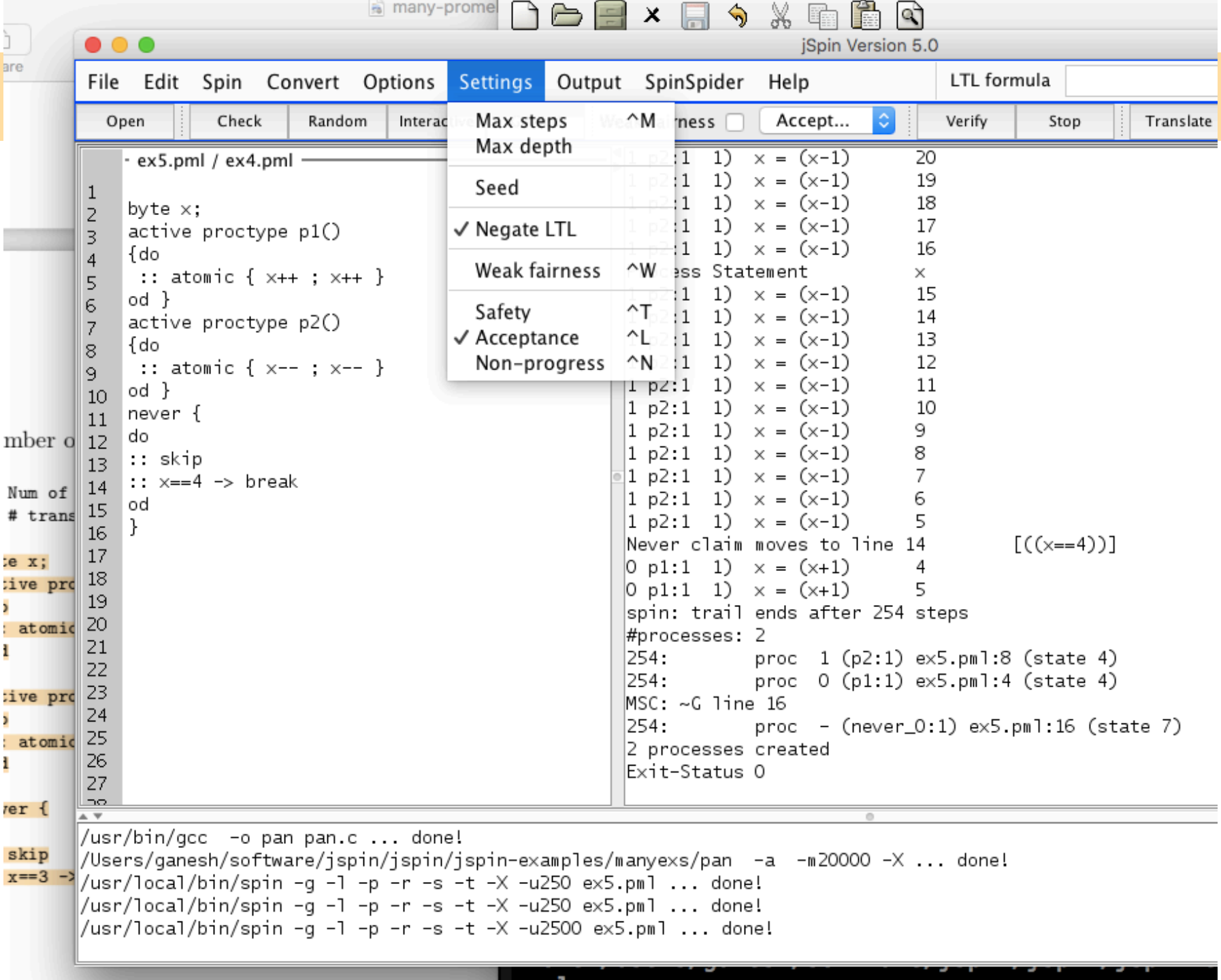
```
/* Num of states stay the same,
   # transitions go up wrt ex4.pr */

byte x;
active proctype p1()
{do
 :: atomic { x++ ; x++ }
 od
}
active proctype p2()
{do
 :: atomic { x-- ; x-- }
 od
}
never {
do
:: skip
:: x==3 -> break
od
}
```

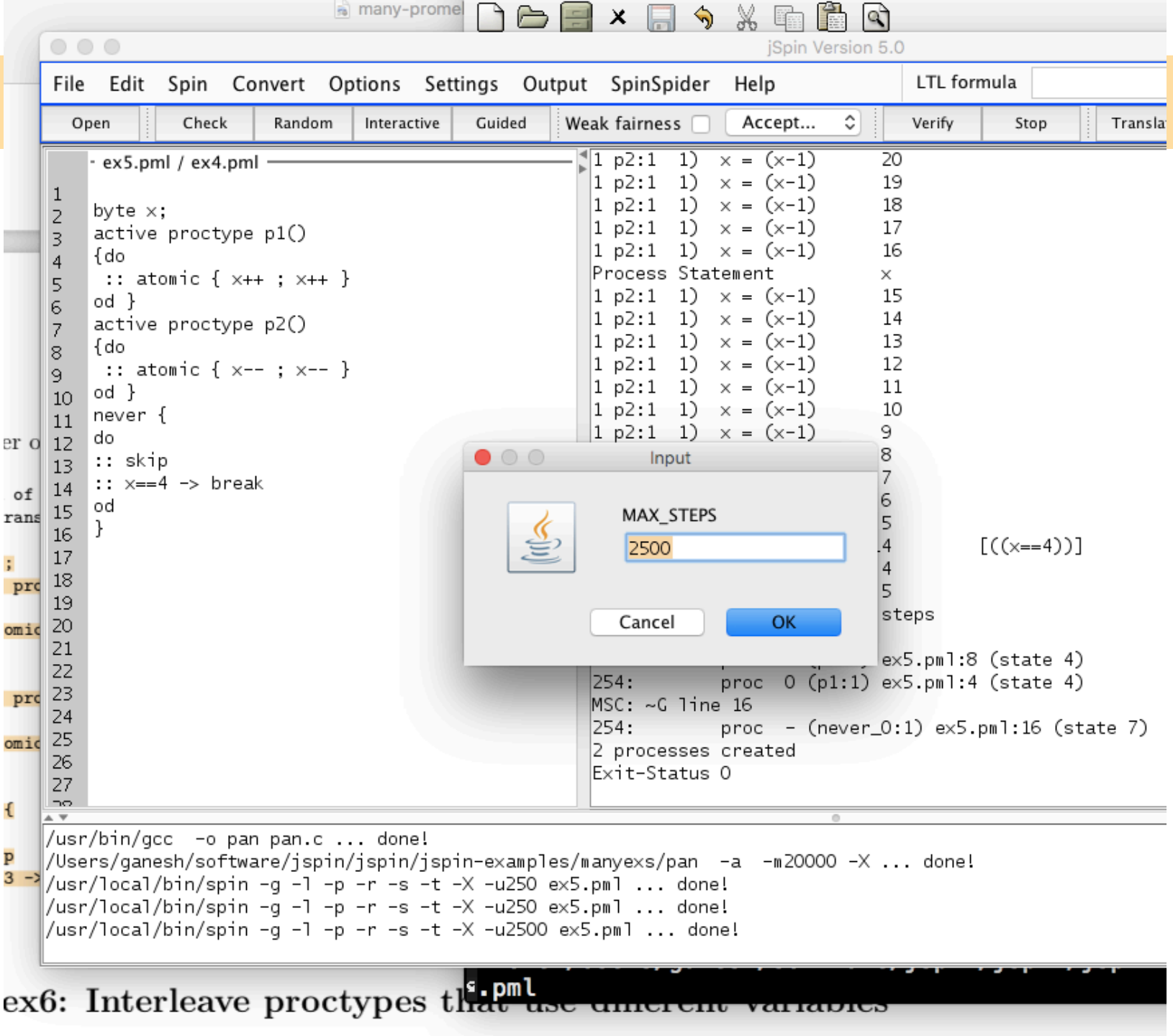
ex5


jSpin Version 5.0
Options
Settings
Output
SpinSpider
Help
LTL formula
Max steps ^M
Max depth
Seed
Negate LTL
Weak fairness ^W
Safety ^T
Acceptance ^L
Non-progress ^N
Accept...
Verify
x++ }
x-- }
Statement
1 p2:1 1) x = (x-1) 20
1 p2:1 1) x = (x-1) 19
1 p2:1 1) x = (x-1) 18
1 p2:1 1) x = (x-1) 17
1 p2:1 1) x = (x-1) 16
1 p2:1 1) x = (x-1) 15
1 p2:1 1) x = (x-1) 14
1 p2:1 1) x = (x-1) 13
1 p2:1 1) x = (x-1) 12
1 p2:1 1) x = (x-1) 11
1 p2:1 1) x = (x-1) 10
1 p2:1 1) x = (x-1) 9

ex5

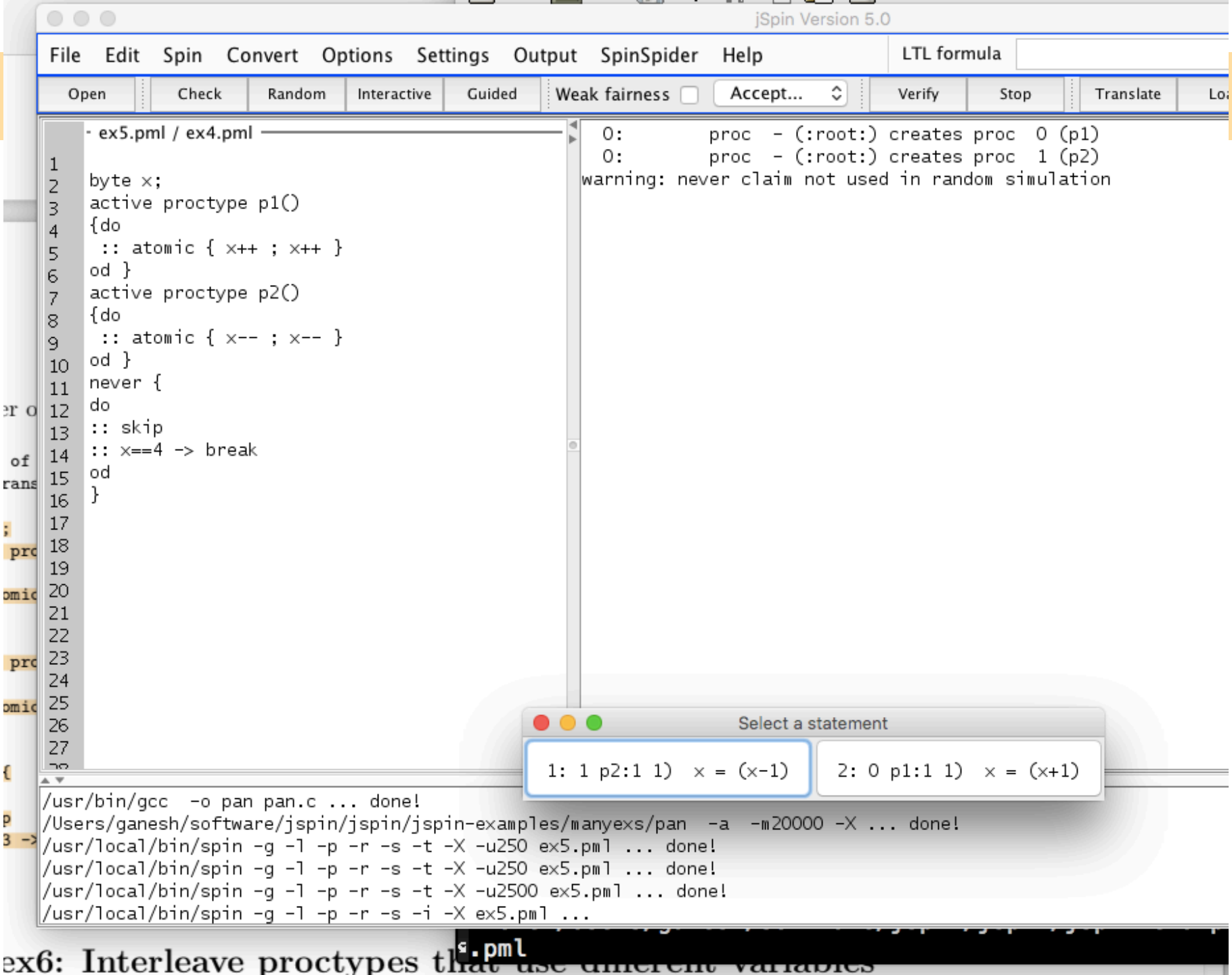

jSpin Version 5.0
File Edit Spin Convert Options Settings Output SpinSpider Help
LTL formula
Open Check Random Interac
Max steps ^M
Max depth
Seed
Negate LTL
Weak fairness ^W
Safety ^T
Acceptance ^L
Non-progress ^N
Accept...
Verify Stop Translate
ex5.pml / ex4.pml
byte x;
active proctype p1()
{do
:: atomic { x++ ; x++ }
od }
active proctype p2()
{do
:: atomic { x-- ; x-- }
od }
never {
do
:: skip
:: x==4 -> break
od
}
1 p2:1 1) x = (x-1) 20
1 p2:1 1) x = (x-1) 19
1 p2:1 1) x = (x-1) 18
1 p2:1 1) x = (x-1) 17
1 p2:1 1) x = (x-1) 16
ess Statement x
1 p2:1 1) x = (x-1) 15
1 p2:1 1) x = (x-1) 14
1 p2:1 1) x = (x-1) 13
1 p2:1 1) x = (x-1) 12
1 p2:1 1) x = (x-1) 11
1 p2:1 1) x = (x-1) 10
1 p2:1 1) x = (x-1) 9
1 p2:1 1) x = (x-1) 8
1 p2:1 1) x = (x-1) 7
1 p2:1 1) x = (x-1) 6
1 p2:1 1) x = (x-1) 5
Never claim moves to line 14 [((x==4))]
0 p1:1 1) x = (x+1) 4
0 p1:1 1) x = (x+1) 5
spin: trail ends after 254 steps
#processes: 2
254: proc 1 (p2:1) ex5.pml:8 (state 4)
254: proc 0 (p1:1) ex5.pml:4 (state 4)
MSC: ~G line 16
254: proc - (never_0:1) ex5.pml:16 (state 7)
2 processes created
Exit-Status 0
/usr/bin/gcc -o pan pan.c ... done!
/Users/ganesh/software/jspin/jspin/jspin-examples/manyexs/pan -a -m20000 -X ... done!
/usr/local/bin/spin -g -l -p -r -s -t -X -u250 ex5.pml ... done!
/usr/local/bin/spin -g -l -p -r -s -t -X -u250 ex5.pml ... done!
/usr/local/bin/spin -g -l -p -r -s -t -X -u2500 ex5.pml ... done!

ex5



ex6: Interleave proctypes that use different variables

ex5



ex6: Interleave proctypes that use different variables

ex6

```
/* Smart reductions saved states */
byte x,y;
active proctype p1()
{do
 :: atomic { x++ ; x++ }
 od
}
active proctype p2()
{do
 :: atomic { y++ ; y++ }
 od
}
never {
do
:: skip
:: (x==3) -> break
od
}
```

ex6

```
/* Smart reductions saved states */
byte x,y;
active proctype p1()
{do
 :: atomic { x++ ; x++ }
 od
}
active proctype p2()
{do
 :: atomic { y++ ; y++ }
 od
}
never {
do
:: skip
:: (x==3) -> break
od
}
```

```
        + Partial Order Reduction
Full statespace search for:
        never claim                   + (never_0)
        assertion violations          + (if within scope of claim)
        acceptance   cycles           - (not selected)
        invalid end states          - (disabled by never claim)
State-vector 36 byte, depth reached 255, ••• errors: 0 •••
      128 states, stored
      129 states, matched
      257 transitions (= stored+matched)
        0 atomic steps
hash conflicts:         0 (resolved)
Stats on memory usage (in Megabytes):
    0.008        equivalent memory usage for states (stored*(State-vector + overhead))
    0.290        actual memory usage for states
  128.000        memory used for hash table (-w24)
    1.068        memory used for DFS stack (-m20000)
  129.264        total actual memory usage
unreached in proctype p1
        ex6.pml:6, state 7, "-end-"
        (1 of 7 states)
unreached in proctype p2
        ex6.pml:10, state 7, "-end-"
        (1 of 7 states)
unreached in claim never_0
        ex6.pml:16, state 7, "-end-"
        (1 of 7 states)
pan: elapsed time 0 seconds
```

ex7

```
/* Reductions don't work */
byte x,y;
active proctype p1()
{do
 :: atomic { x++ ; x++ }
od }
active proctype p2()
{do
 :: atomic { y++ ; y++ }
od }
never {
do
:: skip
:: (x==3)&&(y==2) -> break
od
}
```

ex7
do the
right depth
setting!

```
/* Reductions don't work */
byte x,y;
active proctype p1()
{do
 :: atomic { x++ ; x++ }
od }
active proctype p2()
{do
 :: atomic { y++ ; y++ }
od }
never {
do
:: skip
:: (x==3)&&(y==2) -> break
od
}
```

```
Full statespace search for:
        never claim                 + (never_0)
        assertion violations        + (if within scope of claim)
        acceptance   cycles         - (not selected)
        invalid end states        - (disabled by never claim)
State-vector 36 byte, depth reached 32767, ••• errors: 0 •••
    16384 states, stored
    16385 states, matched
    32769 transitions (= stored+matched)
        0 atomic steps
hash conflicts:         3 (resolved)
Stats on memory usage (in Megabytes):
    1.000        equivalent memory usage for states (stored*(State-vector + overhead))
    1.072        actual memory usage for states
  128.000        memory used for hash table (-w24)
   10.681        memory used for DFS stack (-m200000)
  139.658        total actual memory usage
unreached in proctype p1
        ex7.pml:6, state 7, "-end-"
        (1 of 7 states)
unreached in proctype p2
        ex7.pml:10, state 7, "-end-"
        (1 of 7 states)
unreached in claim never_0
        ex7.pml:16, state 7, "-end-"
        (1 of 7 states)
pan: elapsed time 0.03 seconds
pan: rate 546133.33 states/second
```

ex8
do the
right depth
setting!

```
/* Smart reductions saved states */
byte x,y;
active proctype p1()
{do
 :: atomic { x++ ; x++ }
 od
}
active proctype p2()
{do
 :: atomic { y++ ; y++ }
 od
}
never {
do
:: skip
:: (x==232)&&(y==2) -> break /* observe both vars to introduce
                             * state explosion
                             */
od
}
```

ex8
do the
right depth
setting!

```
1 p2:1  1)  y = (y+1)      253
0 p1:1  1)  x = (x+1)      254
Process Statement          x           y
0 p1:1  1)  x = (x+1)      1           254
1 p2:1  1)  y = (y+1)      2           254
spin: ex8.pml:9, Error: value (256->0 (8)) truncated in assignment
1 p2:1  1)  y = (y+1)      2           255
1 p2:1  1)  y = (y+1)      2           0
1 p2:1  1)  y = (y+1)      2           1
1 p2:1  1)  y = (y+1)      2           2
1 p2:1  1)  y = (y+1)      2           3
1 p2:1  1)  y = (y+1)      2           4
1 p2:1  1)  y = (y+1)      2           5
```

ex9

```
byte pid1, pid2;

proctype p1()
{byte x; /* Also init to 0 */
 do
 :: x++ ; x++
 od
}
proctype p2()
{byte y; /* Also init to 0 */
 do
 :: y++ ; y++
 od
}
init {
 atomic {
 pid1 = run p1();
 pid2 = run p2();
 }
}

never {
do
:: skip
:: (p1:x==2)&&(p2:y==232) -> break
od
}
```

ex10

```
/* Channels */

chan ch = [0] of { byte }; /* rendezvous channel */

active proctype p1()
{byte x; /* local var x init to 0 */
 do
 :: x++ -> ch!x
 od
}
active proctype p2()
{byte y,z;
 do
 :: ch?y -> z++ /* z tries to keep track of the value of x */
 od
}
never {
do
:: skip
:: (p2:y - p2:z) > 1 -> break
od
}
```

ex11

```
/* Channels */

chan ch = [0] of { byte }; /* rendezvous channel */

active proctype p1()
{byte x; /* local var x init to 0 */
 do
 :: x++ -> ch!x /* ; and -> are the same */
 od
}
active proctype p2()
{byte y,z; /* can be named x, but keeping distinct names */
 do
 :: ch?y -> z++ /* z tracks value of x */
 od
}
never {
do
:: skip
:: (p1:x - p2:z) > 1 -> break
od
}
```

ex12

```
/* set depth-bound of DFS to around 20 */

chan ch = [1] of { byte }; /* buffering (non-rendezvous) channel */

active proctype p1()
{byte x; /* local var x init to 0 */
 do
 :: x++ -> ch!x /* ; and -> are the same */
 od
}
active proctype p2()
{byte y,z; /* can be named x, but keeping distinct names */
 do
 :: ch?y -> z++ /* z tracks value of x */
 od
}
never {
do
:: skip
:: (p1:x - p2:z) > 2 -> break
od
}
```

ex12b

ex12b.pml / ex12.pml

```
/* set depth-bound of DFS to around 20 */
chan ch = [1] of { byte }; /* buffering (non-rendezvous) channel */
byte x, z;
active proctype p1()
{
 do
 :: x++ -> ch!x /* ; and -> are the same */
 od
}
active proctype p2()
{byte y; /* can be named x, but keeping distinct names */
 do
 :: ch?y -> z++ /* z tracks value of x */
 od
}
never {
do
:: skip
:: (x - z) > 2 -> break
od;
accept: goto accept
}
```

```
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:1: acceptance cycle (at depth 4082)
pan: wrote ex12b.pml.trail
(Spin Version 6.4.5 -- 1 January 2016)
Warning: Search not completed
        + Partial Order Reduction
Full statespace search for:
        never claim                 + (never_0)
        assertion violations        + (if within scope of claim)
        acceptance   cycles         + (fairness disabled)
        invalid end states        - (disabled by never claim)
State-vector 44 byte, depth reached 6129, ●●● errors: 1 ●●●
     3074 states, stored
      514 states, matched
     3588 transitions (= stored+matched)
        0 atomic steps
hash conflicts:         0 (resolved)
Stats on memory usage (in Megabytes):
    0.211        equivalent memory usage for states
(stored*(State-vector + overhead))
    0.388        actual memory usage for states
  128.000        memory used for hash table (-w24)
 1068.115        memory used for DFS stack (-m20000000)
 1196.408        total actual memory usage
pan: elapsed time 0.05 seconds
pan: rate     61480 states/second
```

ex13

```
active proctype p1()
{byte x;
 do
 :: x = x + 3 /* USER BEWARE: this statement is atomic, unlike in C !! */
 od
}
active proctype p2()
{byte y;
 do
 :: y = y + 5
 od
}
never {
do
:: skip
:: (p1:x == p2:y) -> break
od;
accept: goto accept; /* not needed but looks Buchi */
}
```

ex13

```
active proctype p1()
{byte x;
 do
 :: x = x + 3 /* USER BEWARE: this statement is atomic, unlike in C !! */
 od
}
active proctype p2()
{byte y;
 do
 :: y = y + 5
 od
}
never {
do
:: skip
:: (p1:x == p2:y) -> break
od;
accept: goto accept; /* not needed but looks Buchi */
}
```

http://spinroot.com/spin/Man/Pan.html suggests using -DNOREDUCE
PO reductions not safe with remote references (must be stutter invariant)

ex13

```
active proctype p1()
{byte x;
 do
 :: x = x + 3 /* USER BEWARE: this statement is atomic, unlike in C !! */
 od
}
active proctype p2()
{byte y;
 do
 :: y = y + 5
 od
}
never {
do
:: skip
:: (p1:x == p2:y) -> break
od;
accept: goto accept; /* not needed but looks Buchi */
}
```

ex14

```
mtype = {are_you_free, yes, no, release}
byte progress; /* SPIN initializes all variables to 0 */
proctype phil(chan lf, rf; int philno)
{ do
  :: do
     :: lf!are_you_free ->
        if
        :: lf?yes -> break
        :: lf?no
        fi
     od;
     do
     :: rf!are_you_free ->
        if
        :: rf?yes -> progress = 1 -> progress = 0
                  -> lf!release   -> rf!release -> break
        :: rf?no  -> lf!release   ->  break
        fi
     od
  od
}
proctype fork(chan lp, rp)
{ do
  :: rp?are_you_free -> rp!yes ->
     do
     :: lp?are_you_free -> lp!no
     :: rp?release      -> break
     od
  :: lp?are_you_free -> lp!yes ->
     do
     :: rp?are_you_free -> rp!no
     :: lp?release      -> break
     od
  od
}
init {
   chan c0 = [0] of { mtype }; chan c1 = [0] of { mtype };
   chan c2 = [0] of { mtype }; chan c3 = [0] of { mtype };
   chan c4 = [0] of { mtype }; chan c5 = [0] of { mtype };
   atomic {
     run phil(c5, c0, 0); run fork(c0, c1);
     run phil(c1, c2, 1); run fork(c2, c3);
     run phil(c3, c4, 2); run fork(c4, c5); }
}
never { /* Negation of []<> progress */
 do
 :: skip
 :: (!progress) -> goto accept;
 od;
 accept: (!progress) -> goto accept;
}
```

Lecture 4 : LTL model checking (CEAAT)

Question: What is the theory behind this checking?
Answer: On-the-fly LTL Model Checking using Buchi Automata!

ex12b.pml / ex12.pml

```
/* set depth-bound of DFS to around 20 */
chan ch = [1] of { byte }; /* buffering (non-rendezvous) channel */
byte x, z;
active proctype p1()
{
 do
 :: x++ -> ch!x /* ; and -> are the same */
 od
}
active proctype p2()
{byte y; /* can be named x, but keeping distinct names */
 do
 :: ch?y -> z++ /* z tracks value of x */
 od
}
never {
do
:: skip
:: (x - z) > 2 -> break
od;
accept: goto accept
}
```

```
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:1: acceptance cycle (at depth 4082)
pan: wrote ex12b.pml.trail
(Spin Version 6.4.5 -- 1 January 2016)
Warning: Search not completed
        + Partial Order Reduction
Full statespace search for:
        never claim                 + (never_0)
        assertion violations        + (if within scope of claim)
        acceptance   cycles         + (fairness disabled)
        invalid end states        - (disabled by never claim)
State-vector 44 byte, depth reached 6129, ●●● errors: 1 ●●●
     3074 states, stored
      514 states, matched
     3588 transitions (= stored+matched)
        0 atomic steps
hash conflicts:         0 (resolved)
Stats on memory usage (in Megabytes):
    0.211        equivalent memory usage for states
(stored*(State-vector + overhead))
    0.388        actual memory usage for states
  128.000        memory used for hash table (-w24)
 1068.115        memory used for DFS stack (-m20000000)
 1196.408        total actual memory usage
pan: elapsed time 0.05 seconds
pan: rate     61480 states/second
```

Lecture 4 : LTL model checking (CEAAT)

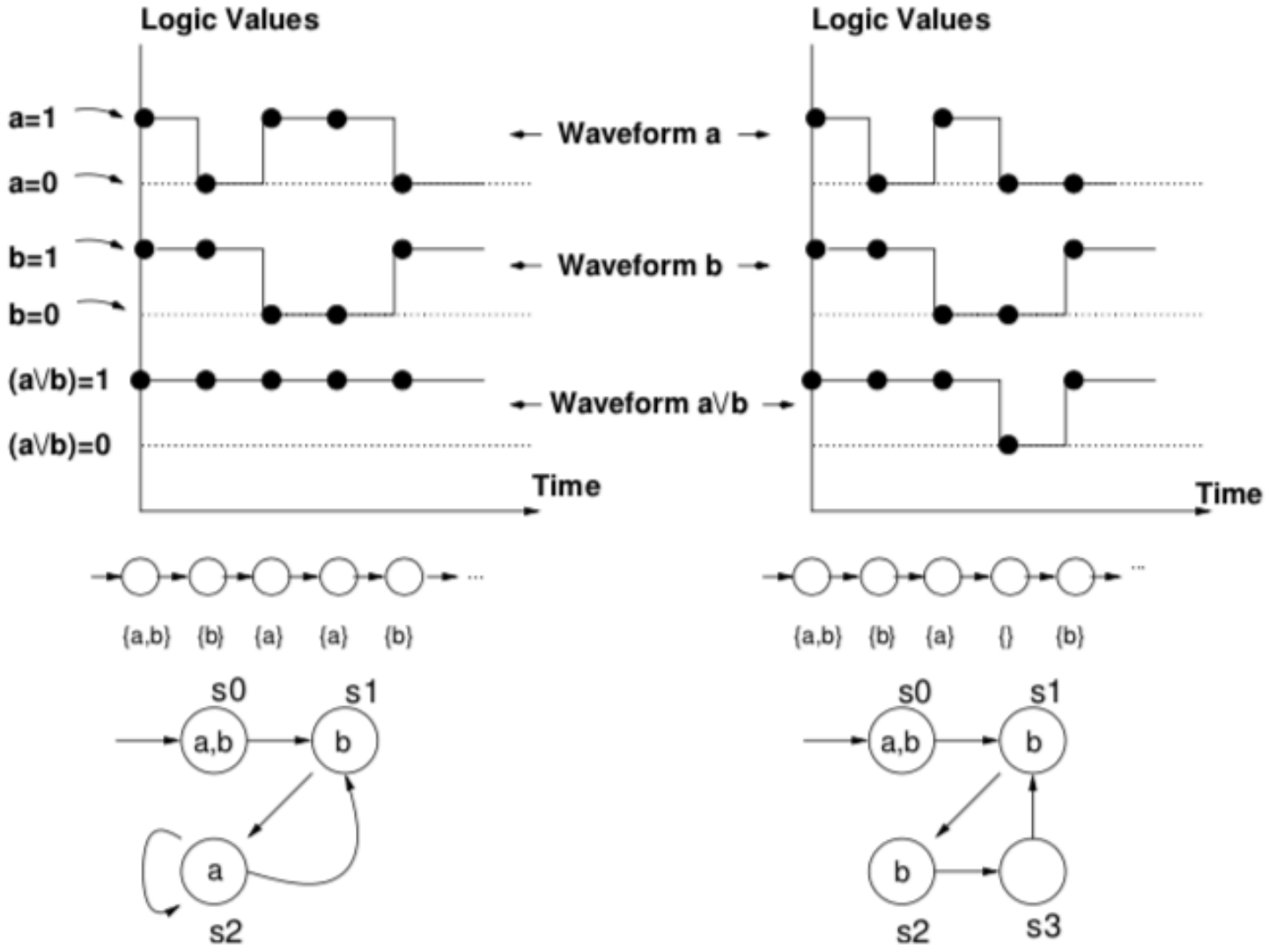


Fig. 22.1. Two Kripke structures and some of their computations. In the Kripke structure on the left, the assertion 'Henceforth $(a \vee b)$' is true.

Lecture 4 : LTL model checking (CEAAT)

22.1.5 LTL syntax

LTL formulas φ are inductively defined as follows, through a context-free grammar:

$\varphi \to x$,	a propositional variable
$\mid \neg\varphi$	negation of an LTL formula
$\mid (\varphi)$	parenthesization
$\mid \varphi_1 \vee \varphi_2$	disjunction
$\mid \mathrm{G}\varphi$	henceforth φ
$\mid \mathrm{F}\varphi$	eventually φ ("future")
$\mid \mathrm{X}\varphi$	next φ
$\mid (\varphi_1 \mathrm{U}\ \varphi_2)$	φ_1 until φ_2
$\mid (\varphi_1 \mathrm{W}\ \varphi_2)$	φ_1 weak-until φ_2

Lecture 4 : LTL model checking (CEAAT)

Here is the inductive definition for the semantics of LTL:

$\sigma \models x$	iff x is true at s_0 (written $s_0(x)$)
$\sigma \models \neg\varphi$	iff $\sigma \not\models \varphi$
$\sigma \models (\varphi)$	iff $\sigma \models \varphi$
$\sigma \models \varphi_1 \vee \varphi_2$	iff $\sigma \models \varphi_1 \vee \sigma \models \varphi_2$
$\sigma \models \mathrm{G}\varphi$	iff $\sigma^i \models \varphi$ for every $i \geq 0$
$\sigma \models \mathrm{F}\varphi$	iff $\sigma^i \models \varphi$ for some $i \geq 0$
$\sigma \models \mathrm{X}\varphi$	iff $\sigma^1 \models \varphi$
$\sigma \models (\varphi_1 \mathrm{U}\ \varphi_2)$	iff $\sigma^k \models \varphi_2$ for some $k \geq 0$ and $\sigma^j \models \varphi_1$ for all $j < k$
$\sigma \models (\varphi_1 \mathrm{W}\ \varphi_2)$	iff $\sigma \models \mathrm{G}\varphi_1 \vee \sigma \models (\varphi_1 \mathrm{U}\ \varphi_2)$

Lecture 4 : LTL model checking (CEAAT)

$$re \rightarrow \emptyset \mid \varepsilon \mid a \in \Sigma \mid (re) \mid re_1 + re_2 \mid re_1 re_2 \mid re^*$$
$$ore \rightarrow re^\omega \mid re\ ore \mid ore_1 + ore_2$$

Lecture 4 : LTL model checking (CEAAT)

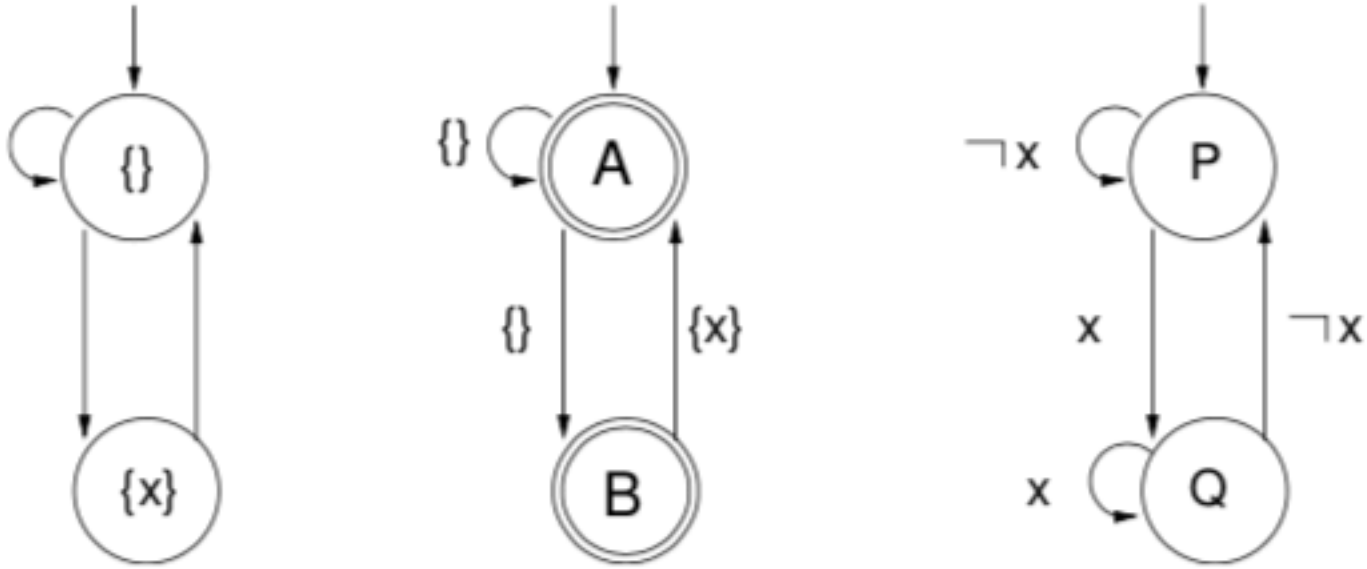


Fig. 23.10. A Kripke structure (left), its corresponding Büchi automata (middle), and a property automaton expressing GFx (right)

Lecture 4 : LTL model checking (CEAAT)

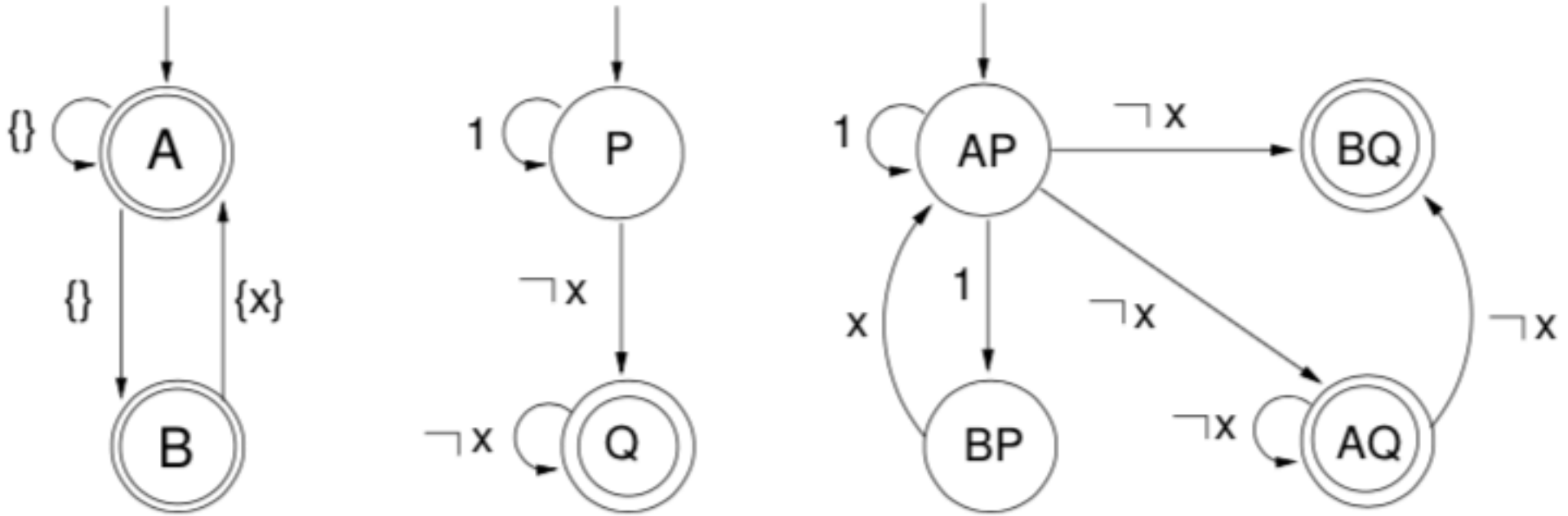


Fig. 23.11. System automaton (left), complemented property automaton (middle), and product automaton (right)

Lecture 4 : LTL model checking (CEAAT)

Notice that the "property automaton" (P) on the right-hand side of Figure 23.10 includes *all* runs that satisfy GFx. Therefore, to determine whether a given Kripke structure ("system" S) satisfies a property P, we can check whether

$$L(S) \subseteq L(P).$$

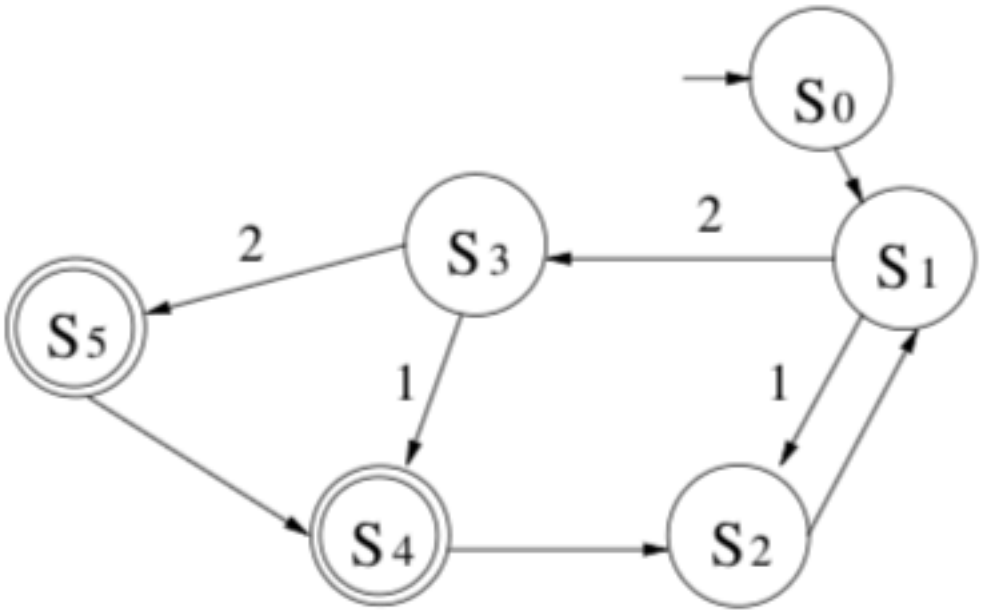
This check is equivalent to

$$L(S) \cap \overline{L(P)} = \emptyset.$$

Lecture 4 : LTL model checking (CEAAT)

$$L(S) \cap \overline{L(P)} \neq \emptyset.$$

Lecture 4 : LTL model checking (CEAAT)



S0	S0	S0	S4	S4
S1	S1	S1	S2	S2
S2	S3	S3	S1	S1
S1	S4	S4	S2	S3
	S2			S4