CS 6110 Software Correctness, Spring 2022 (edited from a previous class; look for bugs!)

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

URL: bit.ly/cs6110s22

Slides for Lec4 : Agenda

- Logic readings
  - Discussions
- FSM material
  - Not familiar to many
    - Not taught as a practical tool in most courses
    - Not taught as the first real formal methods in most courses
      - E.g. context-free parsing is really the first dramatically successful FV tool
        - The horrible bugs in them
          - Fortran missing comma -> spacecraft in a coma ☺
          - The)))*(((rule for parsing!
      - Knuth's article on how bad compilers were in the 1960s !!
        - Youtube version is pretty good (how he got his honeymoon paid for!)
        - https://archive.computerhistory.org/resources/text/Oral_History/Knuth_Don_1/Knuth_Don.oral_history.2007.102658053_all.pdf
  - OpSem basics → Sutter

Slides for Lec4 : Agenda

- How the project discussions are proceeding
  - Notes being taken
  - References being catalogued
  - Not all are at the same point in learning about Logic, Automata, …
    - But they are all important to learn properly!
- We will go through the Bradley/Manna book Ch 1,2
  - Discussion of readings from Ch1
- SPIN Usage
  - Long trace
    - Search depth control

Slides for Lec4 : Agenda

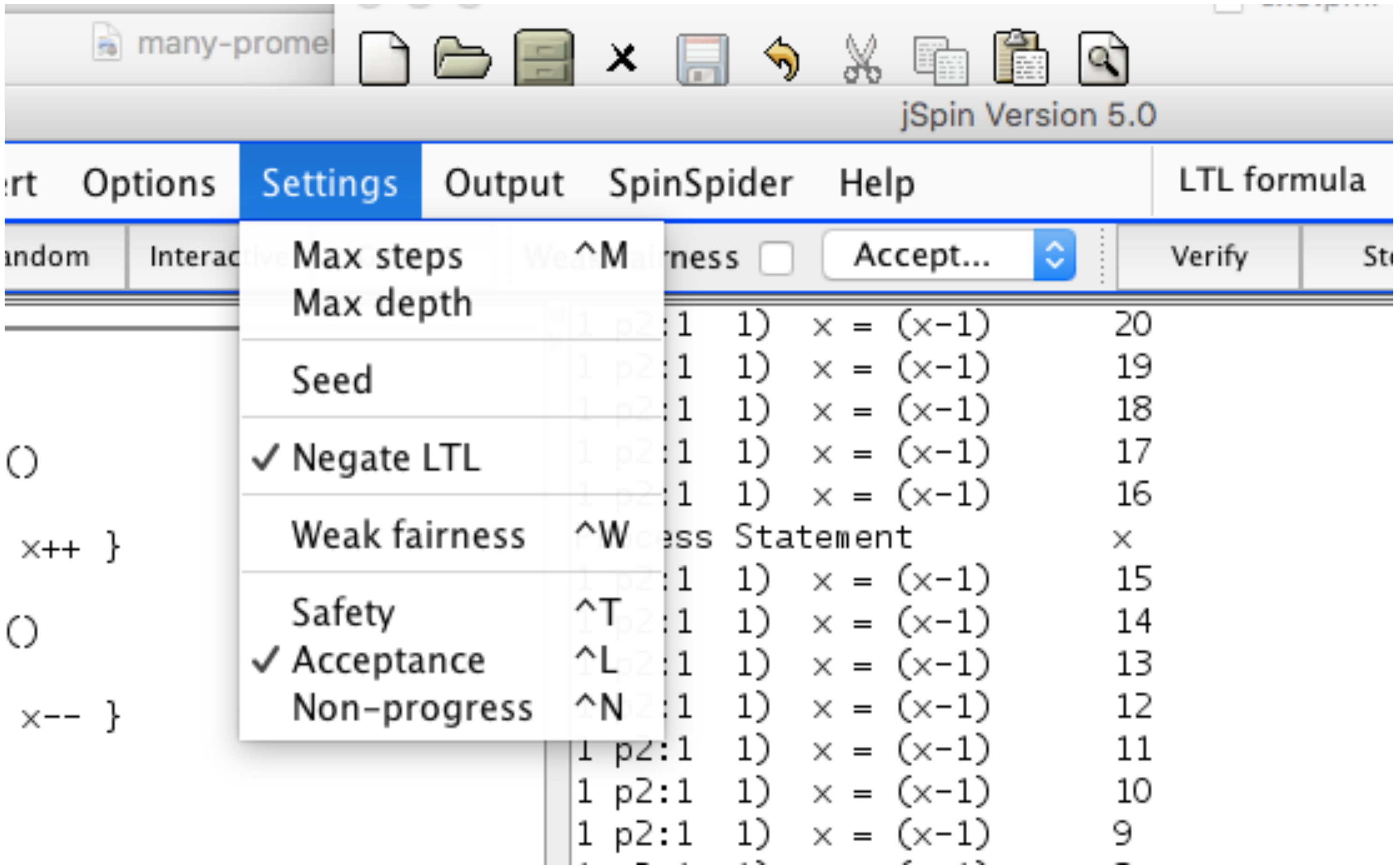
- Readings for next week + Asg-2
  - Posted tomorrow (1/21) evening
  - From Ben-Ari's book

  - From CEATL
    - About NBA != DBA
    - About LTL
    - Nested DFS
    - How we can show LTL validity by creating the right Kripke Structure!
      - Reinforces idea of validity
- Distributed Locking Protocol
  - In Promela
  - In Murphi

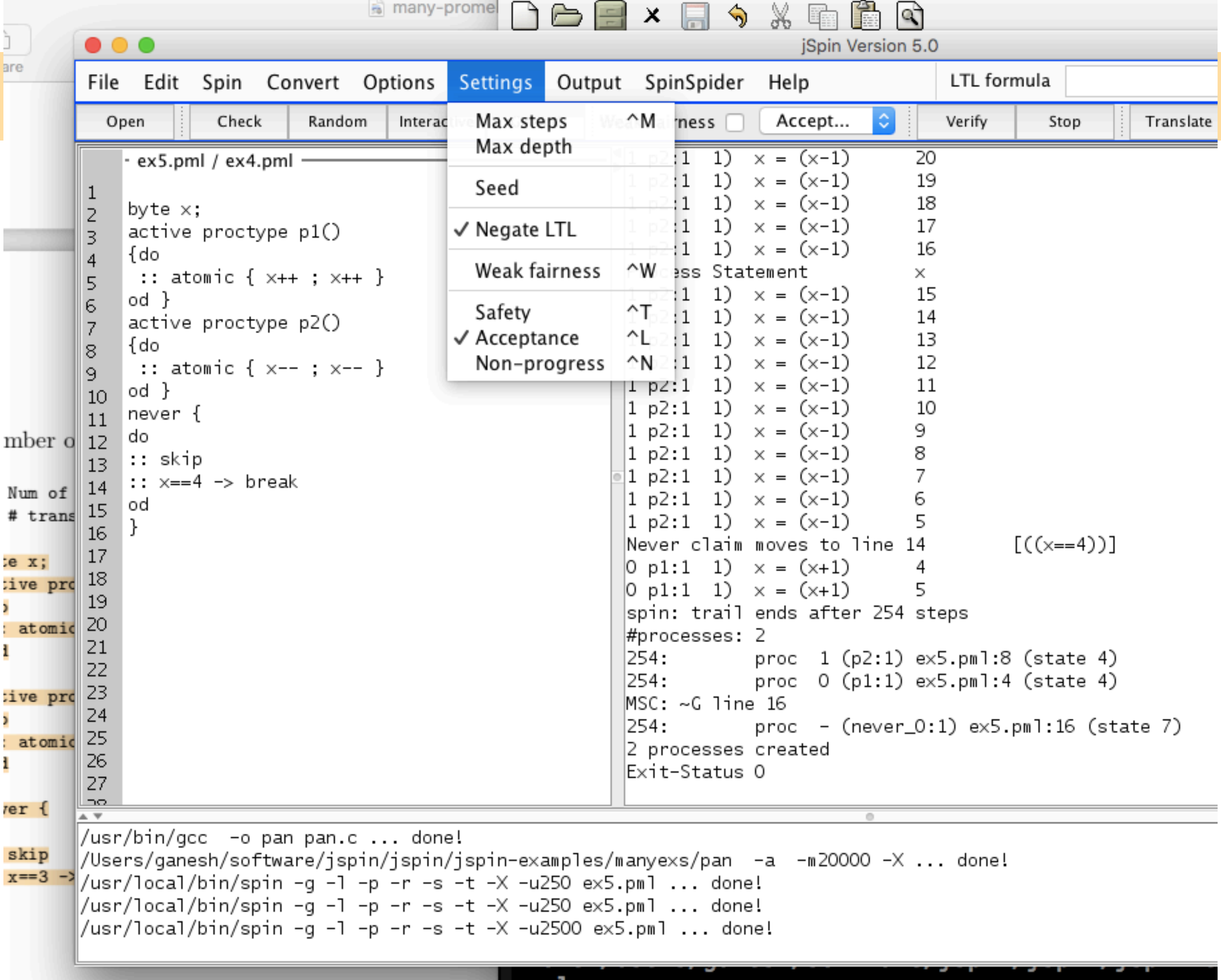
How SPIN helps

- Helps you quickly model “situations”
  - See SPIN-Soldiers + SpinTutorialby Ruys
- Debug your thoughts
- Find flaws in design
- Has a lot of power and will NEVER become irrelevant
  - The more complex a system gets, the less you need to get fancy wrt verification
- (after a 1-month training), you can do this
  - When in doubt, whip up a SPIN model and find bugs (in system or in your head)
- Has worked for large systems
  - Recent correctness workshop -> entire new Pthreads implementation modeled!
- We don’t have the full month to train you
  - This is the last SPIN week … but I’ll give you a REAL protocol verif to read on Thu!

ex5


many-promel
jSpin Version 5.0
Options
Settings
Output
SpinSpider
Help
LTL formula
Interac
Accept...
Verify
Max steps ^M
Max depth
Seed
✓ Negate LTL
Weak fairness ^W
Safety ^T
✓ Acceptance ^L
Non-progress ^N
ness
ess Statement x
1) x = (x-1) 20
1) x = (x-1) 19
1) x = (x-1) 18
1) x = (x-1) 17
1) x = (x-1) 16
1) x = (x-1) 15
1) x = (x-1) 14
1) x = (x-1) 13
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
()
x++ }
()
x-- }

ex5


jSpin Version 5.0
File Edit Spin Convert Options Settings Output SpinSpider Help
LTL formula
Open Check Random Verify Stop Translate
Accept...
Max steps ^M
Max depth
Seed
Negate LTL
Weak fairness ^W
Safety ^T
Acceptance ^L
Non-progress ^N
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
Process 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!

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)

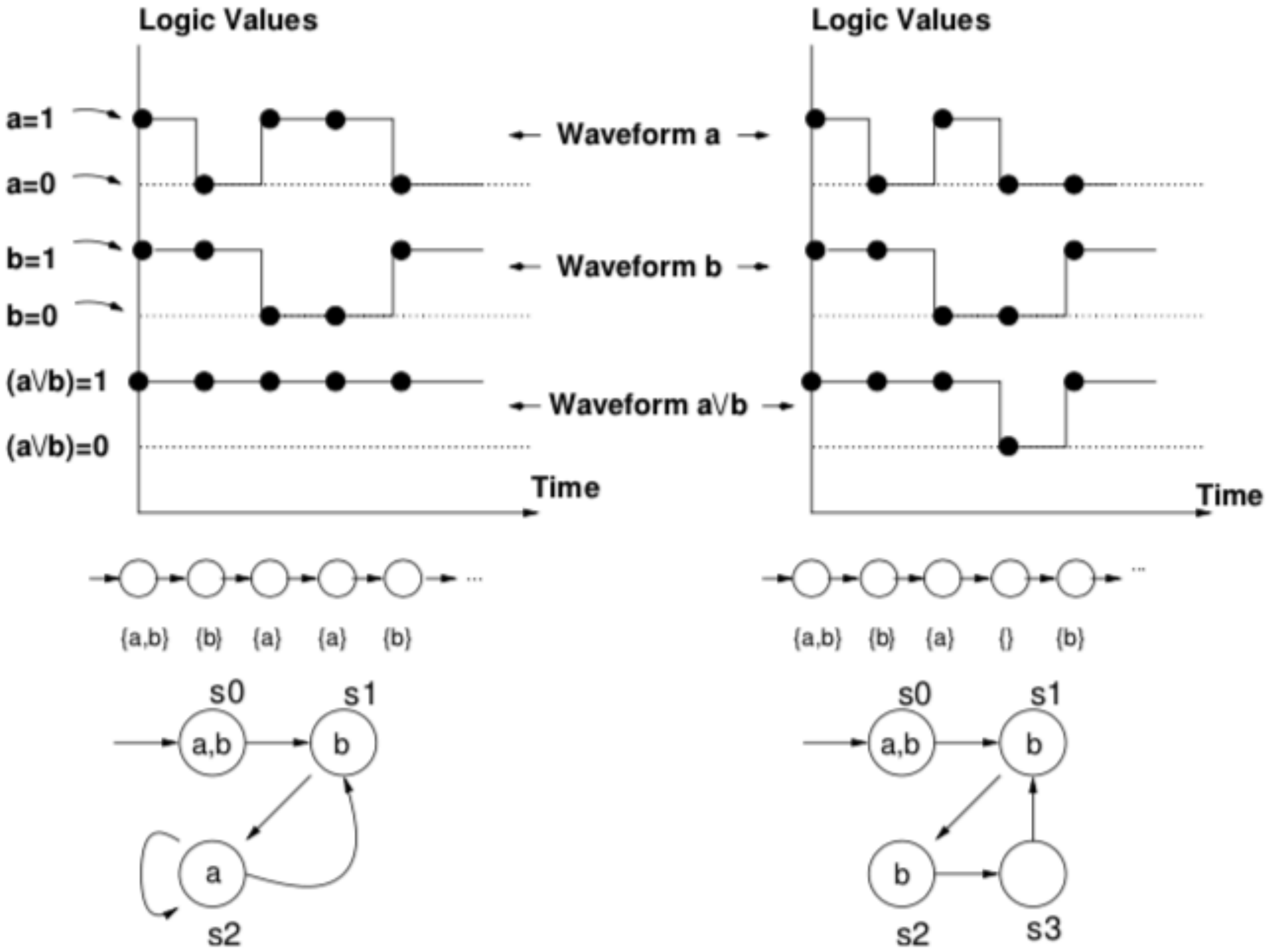


Fig. 22.1. Two Kripke structures and some of their computations. In the Kripke structure on the left, the assertion 'Henceforth $(a \vee b)$' is true.

Lecture 4 : LTL model checking (CEAAT)

22.1.5 LTL syntax

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

$\varphi \rightarrow x,$	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)

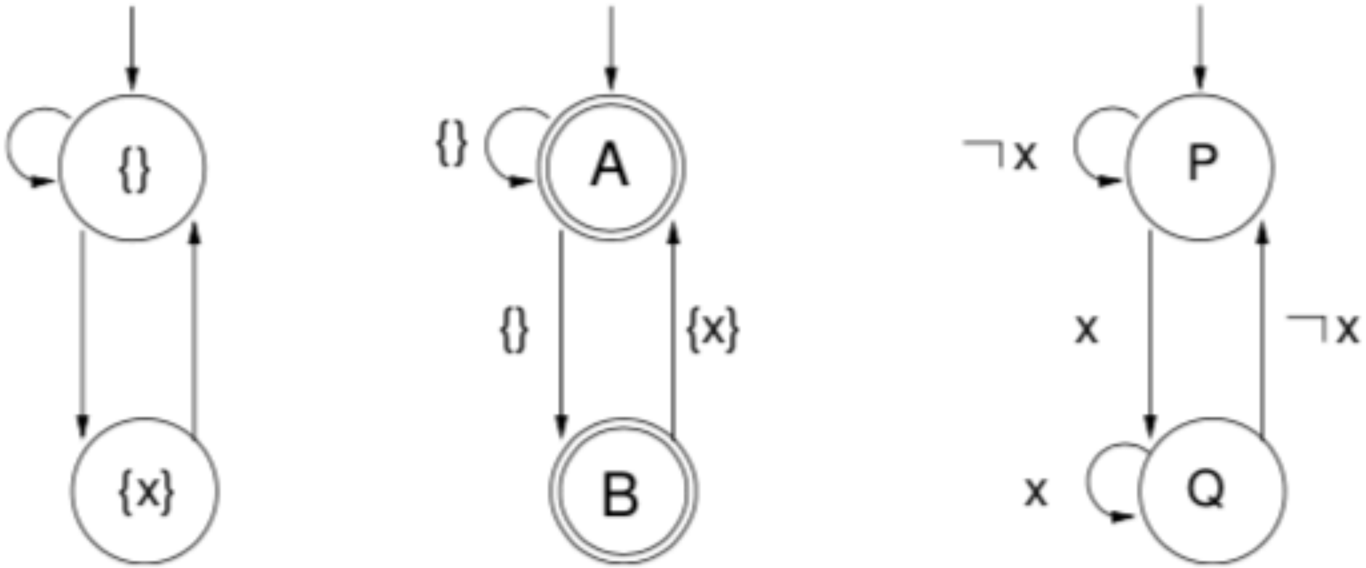


Fig. 23.10. A Kripke structure (left), its corresponding Büchi automata (middle), and a property automaton expressing GFx (right)

Lecture 4 : LTL model checking (CEAAT)

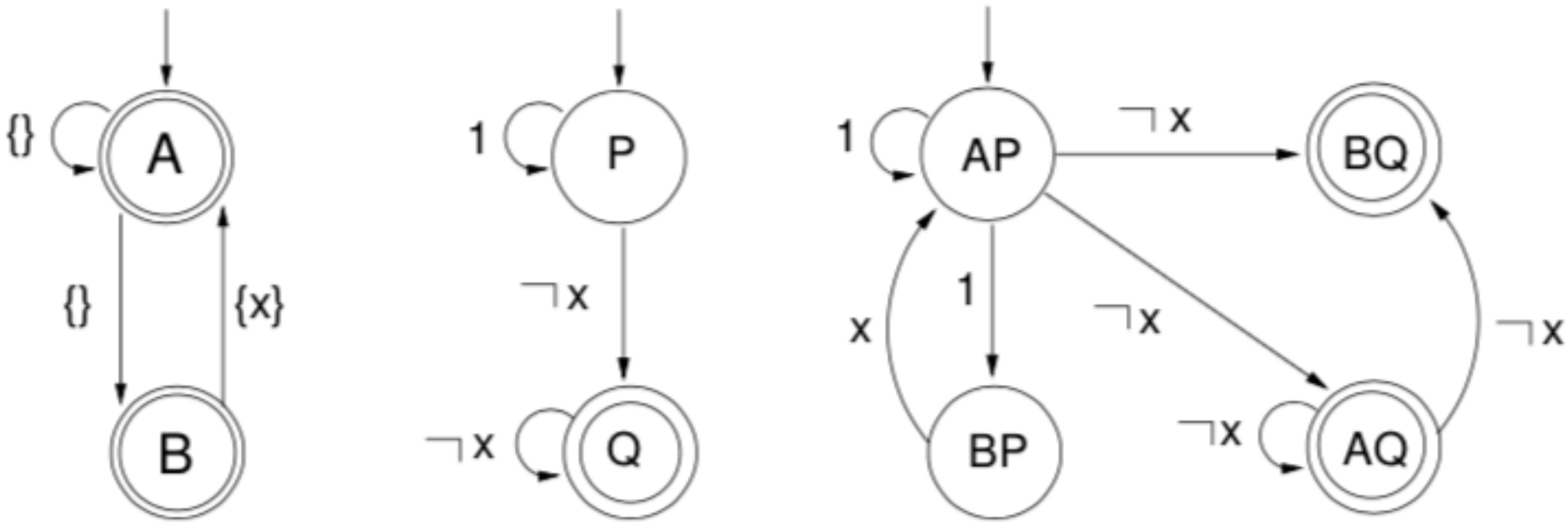


Fig. 23.11. System automaton (left), complemented property automaton (middle), and product automaton (right)

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

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

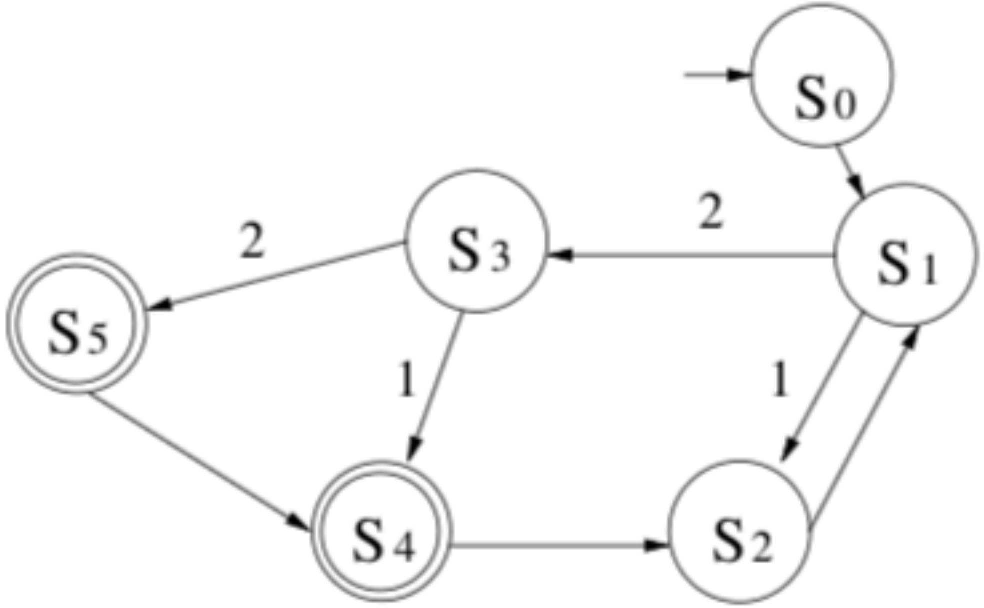
This check is equivalent to

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

Lecture 4 : LTL model checking (CEAAT)

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

Lecture 4 : LTL model checking (CEAAT)



s_0	s_0	s_0	s_4	s_4
s_1	s_1	s_1	s_2	s_2
s_2	s_3	s_3	s_1	s_1
s_1	s_4	s_4	s_2	s_3
	s_2			s_4