

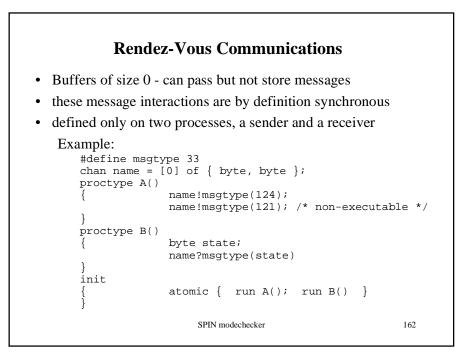
Message Passing

chan qname = [16] of {short} - declaration qname!expr - writing (appending) to the channel qname?expr - reading (from head) of the channel qname!expr1,expr2,expr3 - writing several vars qname?var1,var2,var3 - reading several vars qname!expr1(expr2,expr3) - message type and qname?var1(var2,var3) params qname?cons1,var2,cons2 - can send constants

- · less parameters sent than received others are undefined
- more parameters sent remaining values are lost
- · constants sent must match with constants received

SPIN modechecker

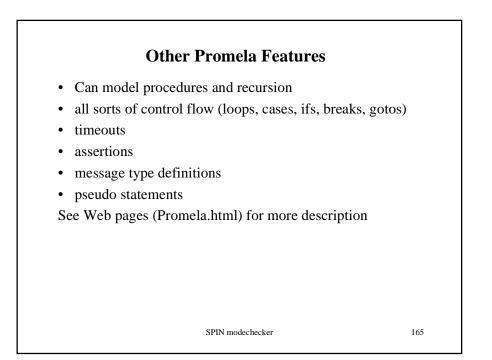
```
Message Passing - Example
   proctype A(chan q1)
     chan q2;
   {
      q1?q2;
      q2!123
   }
   proctype B(chan qforb)
   { int x;
      qforb?x;
      printf(x=%dn'', x)
   }
   init {
      chan qname = [1] of { chan };
      chan qforb = [1] of { int };
      run A(qname);
      run B(qforb);
      qname!qforb
   }
this prints 123
                      SPIN modechecker
                                                      161
```

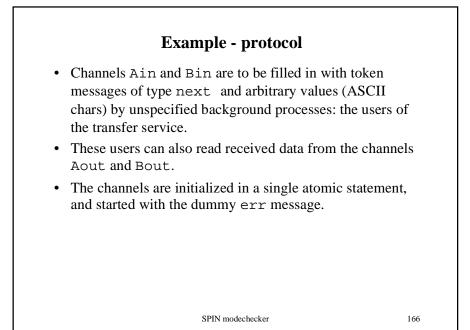


Rendez-Vous Communications (Cont'd) • If channel name has zero buffer capacity: handshake on message msqtype and transfer of value 123 to variable state. The second statement in A will be unexecutable since no matching receive operation in B • If channel name has size 1: process A can complete its first send, but blocks on second since channel is filled. B can retrieve the first message and complete. Then A completes, leaving its last message as a residual in the channel • If channel name has size 2 or more: A can finish its execution before B even starts SPIN modechecker 163

Example using Control-Flow: Dijkstra Semaphore using rendezvous

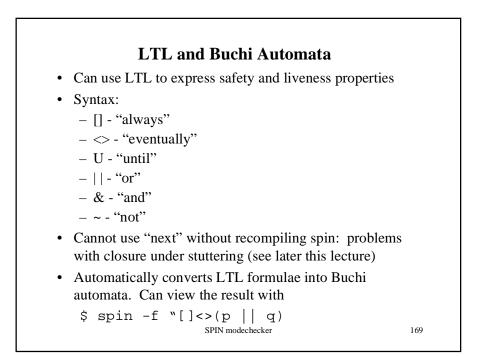
```
#define p
                 0
#define v 1
chan sema = [0] of { bit };
proctype dijkstra()
{ byte count = 1;
   do
   :: (count == 1) -> sema!p; count = 0
:: (count == 0) -> sema? v; count = 1
   od
}
proctype user()
  do
   ::
        sema? p;
         /* crit. sect */
        sema!v;
        /* non-crit. sect. */
   od
init
        run dijkstra(); run user();
{
        run user(); run user()
}
                             SPIN modechecker
```





Ano	ther Example	
mtype = {ack, nak, err	, next, accept};	
proctype transfer(char	in,out,chin,chout)	
{ byte o, I;		
<pre>in?next(o);</pre>		
do		
:: chin?nak(I)	->	
01	<pre>it!accept(I);</pre>	
cł	nout!ack(o)	
:: chin?ack(I)	->	
01	<pre>it!accept(I);</pre>	
ir	n?next(o);	
cł	nout!ack(o)	
:: chin?err(I)	->	
cł	nout!nak(o)	
od		
}		
,		
	SPIN modechecker	167

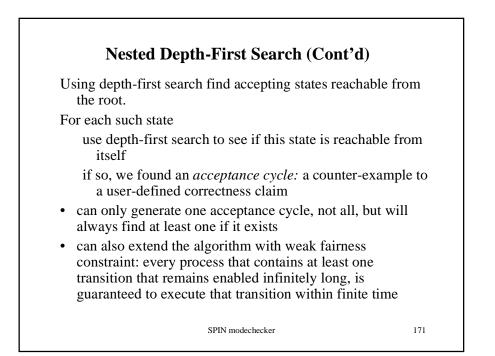
Example (Cont'd)



Nested Depth-First Search

- Problem: need to determine cycles. And the method needs to be compatible with all modes of verification
- Solution (Tarjan) construct strongly-connected components in linear time by adding 2 integers: *dfs*-number and *lowlink*-number (32 bits of storage each because of huge state space)
- Idea: visit each state twice, but storing every state only once. Only 2 bits of overhead instead of 64 by using encoding
- For an accepting cycle to exist in the reachability graph, at least one accepting state must be both reachable from the initial system state (root) and must be reachable from itself

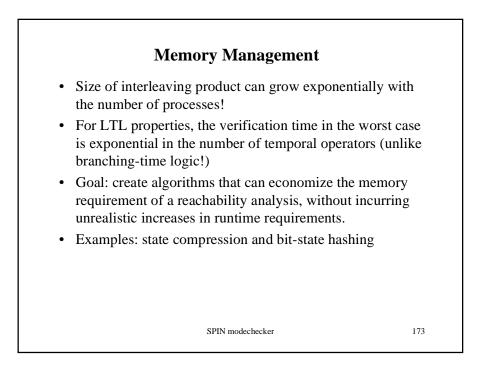
SPIN modechecker

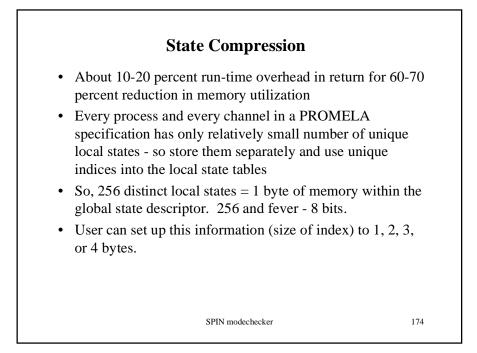


Partial Order Reduction

- Idea: validity of an LTL formula is often insensitive to the order in which concurrent and independently executed events are interleaved in the depth-first search
- Thus can generate a state-space with only representatives of classes of execution sequences that are indistinguishable for a given correctness property
- In a typical case, the reduction in the state space size and in the memory requirements are linear in the size of the model, yielding savings in memory and runtime from 10 to 90 percent.
- This method cannot lead to noticeable increase in memory requirements
- Method not sensitive to decisions about process or variable orderings (unlike BDDs)

SPIN modechecker





Bit-State Hashing	
 Sometimes cannot have exhaustive verifica other techniques stop when they run out of 	,
• With amount of memory M and number of bytes to store each state, the checker exhau after M/S states. <i>Problem coverage</i> is M/(I	sts memory
Example: with 64 bytes of memory to erand total of 2 Mb, we can store 32,768 s	
Bit-state hashing usually does much better	than that.
 Each reachable state is stored using two bit command line: 	s of information
cc -DBITSTATE -o run pan.c	
 Can specify amount of available (non-virtu directly, using -w N option, e.g., -w27 m have 128 Mb of memory. 	
SPIN modechecker	175

Bit-Size Hashing

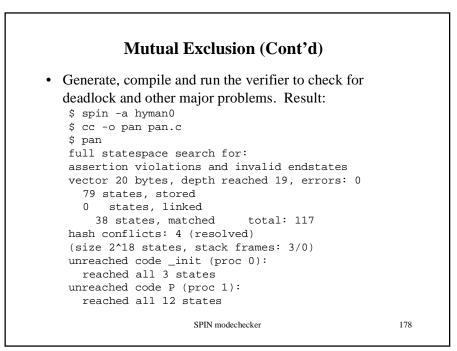
• Exact algorithm could not be determined, but here is an example:

```
$ run
assertion violated (I == ((last_I + 1))
pan: aborted
search interrupted
...
hash factor: 67650.064516
(size 2^22 states, stack frames: 0/5)
```

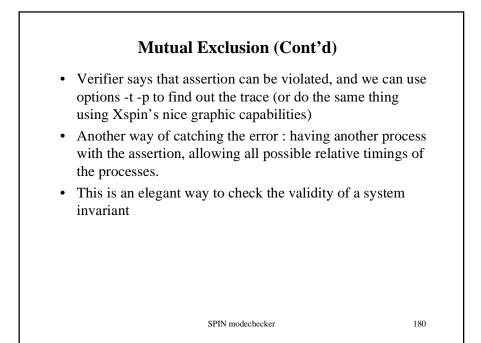
- Hash factor: maximum number of states/actual number
- Maximum number of states is 2²² bytes or about 32 million bits = states
- Hash factor > 100 coverage around 100%
- Hash factor = 1 -> coverage approaches 0%

```
SPIN modechecker
```

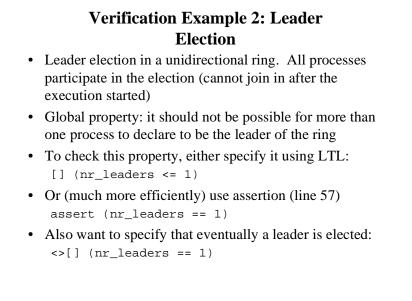
```
Verification example 1: mutual
                     exclusion
1
  bool want[2]; /* Bool array b */
2
   bool turn;
               /* integer
                             k */
3
4
  proctype P(bool I)
5
   {
6
    want[I] = 1;
7
    do
8
    :: (turn != I) ->
9
            (!want[1-I]);
10
             turn = I
11
     :: (turn == I) ->
12
            break
13
     od
      skip; /* critical section */
14
15
     want[I] = 0
16 }
17
18 init { run P(0); run P(1) }
                      SPIN modechecker
                                                     177
```



Mutual exclusion (Cont'd)			
• Want to check mutual exclusion.			
<pre>1 bool want[2]; /* Bool array b */ 2 bool turn; /* integer k */ 3 byte cnt; 4 proctype P(bool I) 5 { 6 want[I] = 1; 7 do 8 :: (turn != I) -> 9 (!want[1-I]); 10 turn = I 11 :: (turn == I) -> 12 break 13 od 14 skip; /* critical section */ 15 cnt = cnt+1; 16 assert(cnt == 1); 17 cnt = cnt-1 18 want[I] = 0 19 } 20 init { run P(0); run P(1) } </pre>			
SPIN modechecker	179		



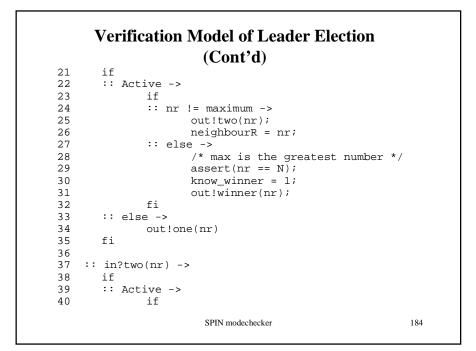
Mutual Exclusion (Cont'd)
<pre>1 bool want[2]; /* Bool array b */ 2 bool turn; /* integer k */ 3 byte cnt; 4 proctype P(bool I) 5 { 6 want[I] = 1; 7 do 8 :: (turn != I) -> 9 (!want[1-I]); 10 turn = I 11 :: (turn == I) -> 12 break 13 od 14 cnt = cnt+1; 15 skip; /* critical section */ 16 cnt = cnt-1; 17 want[I] = 0</pre>
<pre>18 } 19 proctype monitor() 20 { assert (cnt == 0 cnt == 1) } 21 init { run P(0); run P(1); run monitor() }</pre>

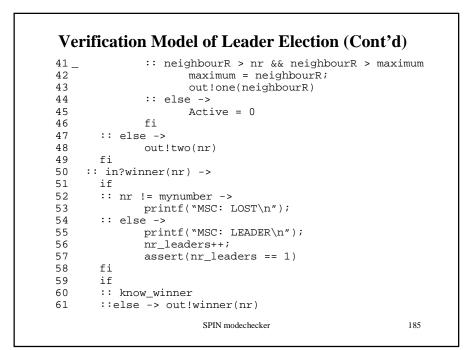


SPIN modechecker

182

Verification Model of Leader Election 1 #define N 5 /* nr of processes */ #define I 3 /* node given the smallest number */ 2 #define L 10 /* size of buffer (>= 2*N) */ 3 4 mtype = { one, two, winner}; /* symb. Msg. Names */ 5 chan q[N] = [L] of {mtype, byte} /* assynch. Chnl */ 6 7 8 byte nr_leaders = 0; /* count the number of process that think they are leader of the ring */ 9 10 proctype node (chan in, out; byte mynumber) 11 { bit Active = 1, know_winner = 0; 12 byte nr, maximum = mynumber, neighbourR; 13 xr in; /* claim exclusive recv access to in */ 14 15 xs out; /* claim exclusive send access to out */ 16 17 printf("MSC: %d\n", mynumber); 18 out!one(mynumber) /* send msg of type one */ 19 end: do 20 :: in?one(nr) -> /* receive msg of type one */ SPIN modechecker 183





```
Verification Model of Leader Election
                       (Cont'd)
      fi;
62
63
      break
64
    od
65 }
66
67
   init {
68
     byte proc;
69
      atomic {/* activate N copies of proc template */
             proc = 1;
70
71
              do
72
              :: proc <= N ->
                  run node (q[proc-1], q[proc % N],
73
74
                     (N+I-proc)% N+1);
                  proc++
75
76
              :: proc > N ->
77
                  break
78
              od
79
       }
   }
80
                       SPIN modechecker
                                                       186
```

