Modelchecking with SPIN

Bernd Finkbeiner – Sven Schewe
Rayna Dimitrova – Lars Kuhtz – Anne Proetzsch

Universität des Saarlandes
Wintersemester 2007/2008

December 13th 2007

Automatic Verification of Dekker's Mutex

```
bit signal[2] = 1;
byte mutex = 0;      /* # procs in the critical section */
byte turn;           /* whose turn is it?  */

proctype proc(byte proc_id) {
   do
   :: 1 -> skip;
   :: signal[proc_id] = 1;
      turn = 1-proc_id;
      signal[1-proc_id] == 0 || turn == proc_id;
      printf("%d enters critical section.\n", proc_id);
      mutex++;
      mutex--;
      printf("%d has left critical section.\n", proc_id);
      signal[proc_id] = 0;
   od
}

init {
   atomic { run proc(0); run proc(1);  }
}
```

Automatic Verification of Dekker's Mutex (Safety)

```
bit signal[2] = 1;
byte mutex = 0;        /* # procs in the critical section */
byte turn;             /* whose turn is it?  */

proctype proc(byte proc_id) {
   do
   :: 1 -> skip;
   :: signal[proc_id] = 1;
      turn = 1-proc_id;
      signal[1-proc_id] == 0 || turn == proc_id;
      printf("%d enters critical section.\n", proc_id);
      mutex++;
      assert(mutex != 2);
      mutex--;
      printf("%d has left critical section.\n", proc_id);
      signal[proc_id] = 0;
   od
}

init {
   atomic { run proc(0); run proc(1);  }
}
```

Automatic Verification of Dekker's Mutex (Safety)

```
bit signal[2] = 1;
byte mutex = 0;       /* # procs in the critical section */
byte turn;            /* whose turn is it?  */

proctype proc(byte proc_id) {
   do
   :: 1 -> skip;
   :: signal[proc_id] = 1;
      turn = 1-proc_id;
      signal[1-proc_id] == 0 || turn == proc_id;
      printf("%d enters critical section.\n", proc_id);
      mutex++;
      mutex--;
      printf("%d has left critical section.\n", proc_id);
      signal[proc_id] = 0;
   od
}

proctype monitor() {
   assert(mutex != 2);
}

init {
   atomic { run proc(0); run proc(1); run monitor(); }
}
```

Automatic Verification of Dekker's Mutex (Liveness)

```
bit signal[2] = 1;
byte mutex = 0;      /* # procs in the critical section */
byte turn;           /* whose turn is it?  */

proctype proc(byte proc_id) {
   do
   :: 1 -> skip;
   :: signal[proc_id] = 1;
      turn = 1-proc_id;
      signal[1-proc_id] == 0 || turn == proc_id;
      printf("%d enters critical section.\n", proc_id);
      mutex++;
      mutex--;
      printf("%d has left critical section.\n", proc_id);
      signal[proc_id] = 0;
   od
}

  proctype monitor() {
        assert( [] ((signal[0] == 1) -> <> (signal[0] == 0)) );
  }

init {
   atomic { run proc(0); run proc(1); run monitor(); }
}
```

Automatic Verification of Dekker's Mutex (Liveness)

```
proctype monitor() {
        assert( [] ((signal[0] == 1) -> <> (signal[0] == 0)) );
}
```

. . . translated into an NBA . . .

```
#define r (signal[0] == 1)
#define g (signal[0] == 0)

never {   /* !([] (r -> <> g)) */
T0_init:
    if
    :: (!  ((g)) && (r)) -> goto accept_S4
    :: (1) -> goto T0_init
    fi ;
accept_S4:
    if
    :: (!  ((g))) -> goto accept_S4
    fi ;
}
```

SPIN

- SPIN: Simple Promela Interpreter
- Promela: Protocol/ Process Meta Languge
- Explicit State Modelchecker for LTL
- Version 1.0: *Holzmann (1991)*
- Implements *Vardi and Wolper (1986)*
- Developed at Bell Labs
- Sourcecode and documentation:

  http://spinroot.com

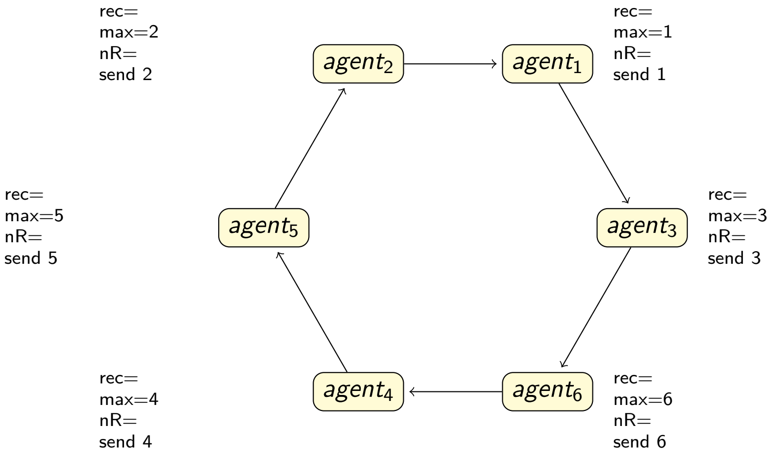
Overview of Promela

- Suitable for concurrent and reactive systems (e.g. Protocols)
- Dynamic process creation
- Explicit atomicity
- Communication via shared memory
- Communication via message passing (asynchronous and synchronous)
- Nondeterministic control
- Guarded execution of statements
- Straight forward encoding of NBA

Only finite data domains

Bound on maximal number of concurrent processes

Example: Leader Election



rec=
max=2
nR=
send 2

$agent_2$

$agent_1$

rec=
max=1
nR=
send 1

rec=
max=5
nR=
send 5

$agent_5$

$agent_3$

rec=
max=3
nR=
send 3

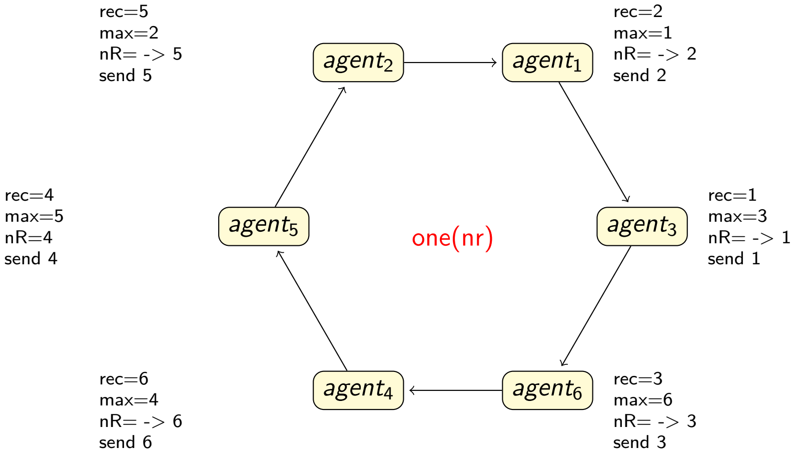
rec=
max=4
nR=
send 4

$agent_4$

$agent_6$

rec=
max=6
nR=
send 6

Example: Leader Election



rec=5
max=2
nR= -> 5
send 5

$agent_2$

$agent_1$

rec=2
max=1
nR= -> 2
send 2

rec=4
max=5
nR=4
send 4

$agent_5$

one(nr)

$agent_3$

rec=1
max=3
nR= -> 1
send 1

rec=6
max=4
nR= -> 6
send 6

$agent_4$

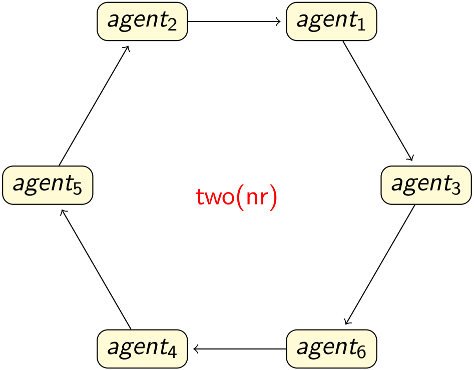
$agent_6$

rec=3
max=6
nR= -> 3
send 3

```
max != nr -> nR = nr, send two(nr)
```

Example: Leader Election

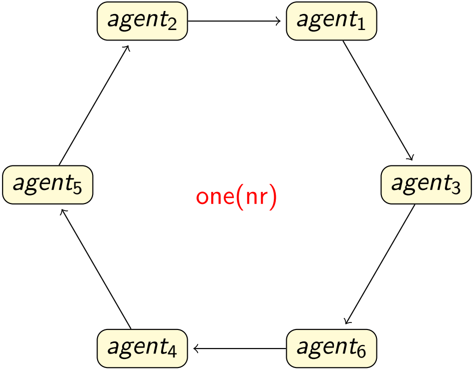
rec=4
max= 2->5
nR=5
send 5



rec=3
max=4->6
nR=6
send 6

```
(nR > nr && nR > max) -> max = nR; send one(nR)
```

Example: Leader Election



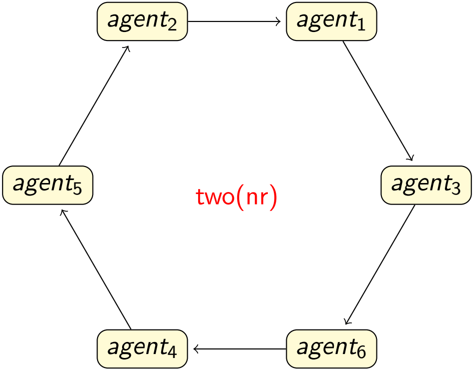
rec=6
max=5
nR=5->6
send 6

rec=5
max=6
nR=6->5
send 5

`max != nr -> neighbourR = nr, send two(nr)`

Example: Leader Election

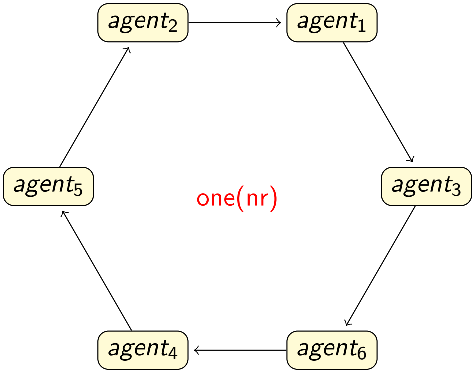
rec=5
max=5->6
nR=6
send 6



```
(nR > nr && nR > max) -> max = nR; send one(nR)
```

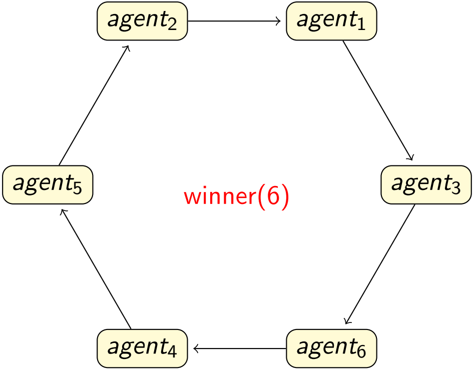
Example: Leader Election

rec=6
max=6
nR=6
send 6



```
max == nr -> know_winner; send winner(nr)
```

Example: Leader Election



```
know_winner -> break
```

```
#define N 5       /* nr of processes (use 5 for demos) */
#define I 3       /* node given the smallest number */
#define L 10      /* size of buffer (>= 2*N) */
mtype = { one, two, winner };
chan q[N] = [L] of { mtype , byte };
byte nr_leaders = 0;
proctype node (chan in, out; byte mynumber) {
   bit Active = 1, know_winner = 0;
   byte nr, maximum = mynumber, neighbourR;
   printf("MSC: %d\n", mynumber);
   out!one(mynumber);
end:    do
   :: in?one(nr) ->
      if
      :: Active ->
         if
         :: nr != maximum ->
            out!two(nr);
            neighbourR = nr
         :: else ->
            assert(nr == N); /* max is greatest number */
            know_winner = 1;
            out!winner,nr;
         fi
      :: else ->
         out!one(nr)
      fi
   :: in?two(nr) ->
      if
      :: Active ->
         if
         :: neighbourR > nr && neighbourR > maximum ->
            maximum = neighbourR;
            out!one(neighbourR)
         :: else ->
            Active = 0
         fi
      :: else ->
         out!two(nr)
      fi
   :: in?winner,nr ->
      if
      :: nr != mynumber ->
         printf("MSC: LOST\n");
      :: else ->
         printf("MSC: LEADER\n");
         nr_leaders++;
         assert(nr_leaders == 1)
      fi ;
      if
      :: know_winner
      :: else -> out!winner,nr
      fi ;
      break
   od
}

init {
   byte proc;
   atomic {
      proc = 1;
      do
      :: proc <= N ->
         run node (q[proc-1], q[proc%N], (N+I-proc)%N+1);
         proc++
      :: proc > N ->
         break
      od
   }
}
```

```
#define N 5       /* nr of processes (use 5 for demos) */
#define I 3       /* node given the smallest number */
#define L 10      /* size of buffer (>= 2*N) */
mtype = { one, two, winner };
chan q[N] = [L] of { mtype , byte };
byte nr_leaders = 0;
proctype node (chan in, out; byte mynumber) {
   bit Active = 1, know_winner = 0;
   byte nr, maximum = mynumber, neighbourR;
   printf("MSC: %d\n", mynumber);
   out!one(mynumber);
end:    do
   :: in?one(nr) ->
      if
      :: Active ->
         if
         :: nr != maximum ->
            out!two(nr);
            neighbourR = nr
         :: else ->
            assert(nr == N); /* max is greatest number */
            know_winner = 1;
            out!winner,nr;
         fi
      :: else ->
         out!one(nr)
      fi
   :: in?two(nr) ->
      if
      :: Active ->
         if
         :: neighbourR > nr && neighbourR > maximum ->
            maximum = neighbourR;
            out!one(neighbourR)
         :: else ->
            Active = 0
         fi
      :: else ->
         out!two(nr)
      fi
```

```
   :: in?winner,nr ->
      if
      :: nr != mynumber ->
         printf("MSC: LOST\n");
      :: else ->
         printf("MSC: LEADER\n");
         nr_leaders++;
         assert(nr_leaders == 1)
      fi ;
      if
      :: know_winner
      :: else -> out!winner,nr
      fi ;
      break
   od
}

init {
   byte proc;
   atomic {
      proc = 1;
      do
      :: proc <= N ->
         run node (q[proc-1], q[proc%N], (N+I-proc)%N+1);
         proc++
      :: proc > N ->
         break
      od
   }
}
```

Promela: (Global) Declarations

```
#define N 5         /* nr of processes (use 5 for demos) */
#define I 3         /* node given the smallest number */
#define L 10        /* size of buffer (>= 2*N) */

mtype = { one, two, winner };
chan q[N] = [L] of { mtype , byte };
byte nr_leaders = 0;
```

- basic types: `bool`, `bit`, `short`, `int`, `byte`, `unsigned`, `pid`
- `mtype`: kind of C-enum
- `typedef`: like C-struct
- arrays of constant length
- `chan` (buffered) channel of size and type
- `proctype`: (parameterized) process type

Channels of size 0 enforce synchronous communication
Code is preprocessed with C-preprocessor

```
#define N 5       /* nr of processes (use 5 for demos) */
#define I 3       /* node given the smallest number */
#define L 10      /* size of buffer (>= 2*N) */
mtype = { one, two, winner };
chan q[N] = [L] of { mtype , byte };
byte nr_leaders = 0;
proctype node (chan in, out; byte mynumber) {
   bit Active = 1, know_winner = 0;
   byte nr, maximum = mynumber, neighbourR;
   printf("MSC: %d\n", mynumber);
   out!one(mynumber);
end:    do
   :: in?one(nr) ->
      if
      :: Active ->
         if
         :: nr != maximum ->
            out!two(nr);
            neighbourR = nr
         :: else ->
            assert(nr == N); /* max is greatest number */
            know_winner = 1;
            out!winner,nr;
         fi
      :: else ->
         out!one(nr)
      fi
   :: in?two(nr) ->
      if
      :: Active ->
         if
         :: neighbourR > nr && neighbourR > maximum ->
            maximum = neighbourR;
            out!one(neighbourR)
         :: else ->
            Active = 0
         fi
      :: else ->
         out!two(nr)
      fi
   :: in?winner,nr ->
      if
      :: nr != mynumber ->
         printf("MSC: LOST\n");
      :: else ->
         printf("MSC: LEADER\n");
         nr_leaders++;
         assert(nr_leaders == 1)
      fi ;
      if
      :: know_winner
      :: else -> out!winner,nr
      fi ;
      break
   od
}

init {
   byte proc;
   atomic {
      proc = 1;
      do
      :: proc <= N ->
         run node (q[proc-1], q[proc%N], (N+I-proc)%N+1);
         proc++
      :: proc > N ->
         break
      od
   }
}
```

Promela: Process Declarations, Statements, Channels

```
proctype node (chan in, out; byte mynumber) {
   bit Active = 1, know_winner = 0;
   byte nr, maximum = mynumber, neighbourR;
   printf("MSC: %d\n", mynumber);
   out!one(mynumber);
end:
   do
   :: in?one(nr) -> skip;
   :: in?two(nr) -> skip;
   od
```

- expressions: like in C
- statements: `skip`, `goto`, `fprint`, `=`, `++`, `--`, any expression
- expr is enabled if it does not evaluate to 0

- `out!args`: put args into channel out
- `in?args`: match args in channel `in`
- `in?[args]`: side effect free test of channel

A receive statement blocks if the match fails

```
#define N 5       /* nr of processes (use 5 for demos) */
#define I 3       /* node given the smallest number */
#define L 10      /* size of buffer (>= 2*N) */
mtype = { one, two, winner };
chan q[N] = [L] of { mtype , byte };
byte nr_leaders = 0;
proctype node (chan in, out; byte mynumber) {
   bit Active = 1, know_winner = 0;
   byte nr, maximum = mynumber, neighbourR;
   printf("MSC: %d\n", mynumber);
   out!one(mynumber);
end:    do
   :: in?one(nr) ->
      if
      :: Active ->
         if
         :: nr != maximum ->
            out!two(nr);
            neighbourR = nr
         :: else ->
            assert(nr == N); /* max is greatest number */
            know_winner = 1;
            out!winner,nr;
         fi
      :: else ->
         out!one(nr)
      fi
   :: in?two(nr) ->
      if
      :: Active ->
         if
         :: neighbourR > nr && neighbourR > maximum ->
            maximum = neighbourR;
            out!one(neighbourR)
         :: else ->
            Active = 0
         fi
      :: else ->
         out!two(nr)
      fi
```

```
   :: in?winner,nr ->
      if
      :: nr != mynumber ->
         printf("MSC: LOST\n");
      :: else ->
         printf("MSC: LEADER\n");
         nr_leaders++;
         assert(nr_leaders == 1)
      fi ;
      if
      :: know_winner
      :: else -> out!winner,nr
      fi ;
      break
   od
}

init {
   byte proc;
   atomic {
      proc = 1;
      do
      :: proc <= N ->
         run node (q[proc-1], q[proc%N], (N+I-proc)%N+1);
         proc++
      :: proc > N ->
         break
      od
   }
}
```

Promela: Guarded Commands

```
:: in?one(nr) ->
   if
   :: Active ->
      if
      :: nr != maximum ->
         out!two(nr);
         neighbourR = nr
      :: else ->
         assert(nr == N); /* max is greatest number */
         know_winner = 1;
         out!winner,nr;
      fi
   :: else ->
      out!one(nr)
   fi
```

- `if :: stmts :: ... fi`: non-deterministic choice
- `do :: stmts :: ... od`: non-deterministic choice + loop
- guarded command: `expr -> statement; ...`
- execution blocks if no statement is enabled (no idling)
- `->` is synonym for `;`

only exit from a loop: `break`

```
#define N 5       /* nr of processes (use 5 for demos) */
#define I 3       /* node given the smallest number */
#define L 10      /* size of buffer (>= 2*N) */
mtype = { one, two, winner };
chan q[N] = [L] of { mtype , byte };
byte nr_leaders = 0;
proctype node (chan in, out; byte mynumber) {
   bit Active = 1, know_winner = 0;
   byte nr, maximum = mynumber, neighbourR;
   printf("MSC: %d\n", mynumber);
   out!one(mynumber);
end:    do
   :: in?one(nr) ->
      if
      :: Active ->
         if
         :: nr != maximum ->
            out!two(nr);
            neighbourR = nr
         :: else ->
            assert(nr == N); /* max is greatest number */
            know_winner = 1;
            out!winner,nr;
         fi
      :: else ->
         out!one(nr)
      fi
   :: in?two(nr) ->
      if
      :: Active ->
         if
         :: neighbourR > nr && neighbourR > maximum ->
            maximum = neighbourR;
            out!one(neighbourR)
         :: else ->
            Active = 0
         fi
      :: else ->
         out!two(nr)
      fi
   :: in?winner,nr ->
      if
      :: nr != mynumber ->
         printf("MSC: LOST\n");
      :: else ->
         printf("MSC: LEADER\n");
         nr_leaders++;
         assert(nr_leaders == 1)
      fi ;
      if
      :: know_winner
      :: else -> out!winner,nr
      fi ;
      break
   od
}

init {
   byte proc;
   atomic {
      proc = 1;
      do
      :: proc <= N ->
         run node (q[proc-1], q[proc%N], (N+I-proc)%N+1);
         proc++
      :: proc > N ->
         break
      od
   }
}
```

```
:: in?two(nr) ->
   if
   :: Active ->
      if
      :: neighbourR > nr && neighbourR > maximum ->
         maximum = neighbourR;
         out!one(neighbourR)
      :: else ->
         Active = 0
      fi
   :: else ->
      out!two(nr)
   fi
```

```
#define N 5       /* nr of processes (use 5 for demos) */
#define I 3       /* node given the smallest number */
#define L 10      /* size of buffer (>= 2*N) */
mtype = { one, two, winner };
chan q[N] = [L] of { mtype , byte };
byte nr_leaders = 0;
proctype node (chan in, out; byte mynumber) {
   bit Active = 1, know_winner = 0;
   byte nr, maximum = mynumber, neighbourR;
   printf("MSC: %d\n", mynumber);
   out!one(mynumber);
end:    do
   :: in?one(nr) ->
      if
      :: Active ->
         if
         :: nr != maximum ->
            out!two(nr);
            neighbourR = nr
         :: else ->
            assert(nr == N); /* max is greatest number */
            know_winner = 1;
            out!winner,nr;
         fi
      :: else ->
         out!one(nr)
      fi
   :: in?two(nr) ->
      if
      :: Active ->
         if
         :: neighbourR > nr && neighbourR > maximum ->
            maximum = neighbourR;
            out!one(neighbourR)
         :: else ->
            Active = 0
         fi
      :: else ->
         out!two(nr)
      fi
```

```
   :: in?winner,nr ->
      if
      :: nr != mynumber ->
         printf("MSC: LOST\n");
      :: else ->
         printf("MSC: LEADER\n");
         nr_leaders++;
         assert(nr_leaders == 1)
      fi ;
      if
      :: know_winner
      :: else -> out!winner,nr
      fi ;
      break
   od
}

init {
   byte proc;
   atomic {
      proc = 1;
      do
      :: proc <= N ->
         run node (q[proc-1], q[proc%N], (N+I-proc)%N+1);
         proc++
      :: proc > N ->
         break
      od
   }
}
```

Promela: Assertions

```
:: in?winner,nr ->
   if
   :: nr != mynumber ->
      printf("MSC: LOST\n");
   :: else ->
      printf("MSC: LEADER\n");
      nr_leaders++;
      assert(nr_leaders == 1)
   fi ;
   if
   :: know_winner
   :: else -> out!winner,nr
   fi ;
   break;
```

- `assert(expr)`: runtime error if expr evaluates to false
- `xr in`: only current process receives from in
- `xs out`: only current process sends on out

```
#define N 5       /* nr of processes (use 5 for demos) */
#define I 3       /* node given the smallest number */
#define L 10      /* size of buffer (>= 2*N) */
mtype = { one, two, winner };
chan q[N] = [L] of { mtype , byte };
byte nr_leaders = 0;
proctype node (chan in, out; byte mynumber) {
   bit Active = 1, know_winner = 0;
   byte nr, maximum = mynumber, neighbourR;
   printf("MSC: %d\n", mynumber);
   out!one(mynumber);
end:    do
   :: in?one(nr) ->
      if
      :: Active ->
         if
         :: nr != maximum ->
            out!two(nr);
            neighbourR = nr
         :: else ->
            assert(nr == N); /* max is greatest number */
            know_winner = 1;
            out!winner,nr;
         fi
      :: else ->
         out!one(nr)
      fi
   :: in?two(nr) ->
      if
      :: Active ->
         if
         :: neighbourR > nr && neighbourR > maximum ->
            maximum = neighbourR;
            out!one(neighbourR)
         :: else ->
            Active = 0
         fi
      :: else ->
         out!two(nr)
      fi
   :: in?winner,nr ->
      if
      :: nr != mynumber ->
         printf("MSC: LOST\n");
      :: else ->
         printf("MSC: LEADER\n");
         nr_leaders++;
         assert(nr_leaders == 1)
      fi ;
      if
      :: know_winner
      :: else -> out!winner,nr
      fi ;
      break
   od
}
```

```
init {
   byte proc;
   atomic {
      proc = 1;
      do
      :: proc <= N ->
         run node (q[proc-1], q[proc%N], (N+I-proc)%N+1);
         proc++
      :: proc > N ->
         break
      od
   }
}
```

Promela: Atomicity, Processes Types

```
init {
   byte proc;
   atomic {
      proc = 1;
      do
      :: proc <= N ->
         run node (q[proc-1], q[proc%N], (N+I-proc)%N+1);
         proc++
      :: proc > N ->
         break
      od
   }
}
```

- `atomic{statements}`: execution of statements if not interrupted
- `run proc(args)`: create process
- special processes: `init`, `never`

Number of processes is bounded (default: 255)

Proving Assertions

- inline assertions

```
   :: in?winner,nr ->
      if
      :: nr != mynumber ->
         printf("MSC: LOST\n");
      :: else ->
         printf("MSC: LEADER\n");
         nr_leaders++;
         assert(nr_leaders == 1)
      fi ;
      if
      :: know_winner
      :: else -> out!winner,nr
      fi ;
      break
   od
}
```

- run a monitor process

```
proctype monitor(){
   assert( nr_leaders <= 1 )
}
```

Proving Temporal Properties

- specification logic: LTL
- translate: LTL -> never-claim
- let SPIN search for accepting cycles