Verification

Lecture 11

Martin Zimmermann

Plan for today

- SPIN

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
}
```

```
ltl prop {  [] ((signal[0] == 1) -> <> (signal[0] == 0)) } ;
```

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

Automatic Verification of Dekker's Mutex (Liveness)

```
ltl prop {  [] ((signal[0] == 1) -> <> (signal[0] == 0)) } ;
```

. . . translated into an NBA . . .

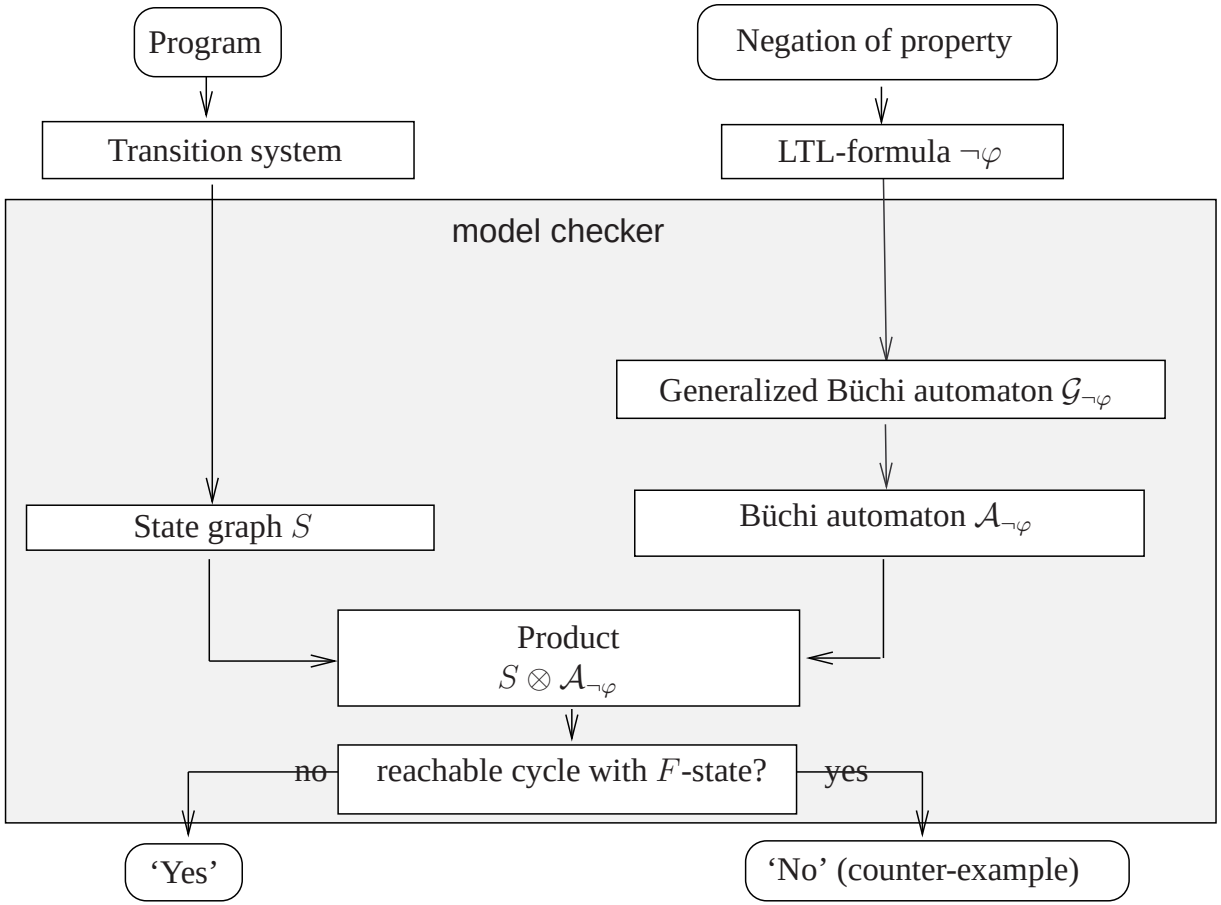
```
#define r (signal[0] == 1)
#define g (signal[0] == 0)

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 Model Checker for LTL
- Version 1.0: *Holzmann (1991)*
- Implements *Vardi and Wolper (1986)*
- Developed at Bell Labs
- Sourcecode and documentation:

http://spinroot.com

Recap: LTL Model Checking



Industrial Applications

Flood Control: verification of the control algorithms for the flood control barrier near Rotterdam

Call Processing: logic verification of the call processing software for the PathStar switch from Lucent Technologies

Mission Critical Software: algorithms for NASA missions including Deep Space 1, Cassini, the Mars Exploration Rovers, Deep Impact

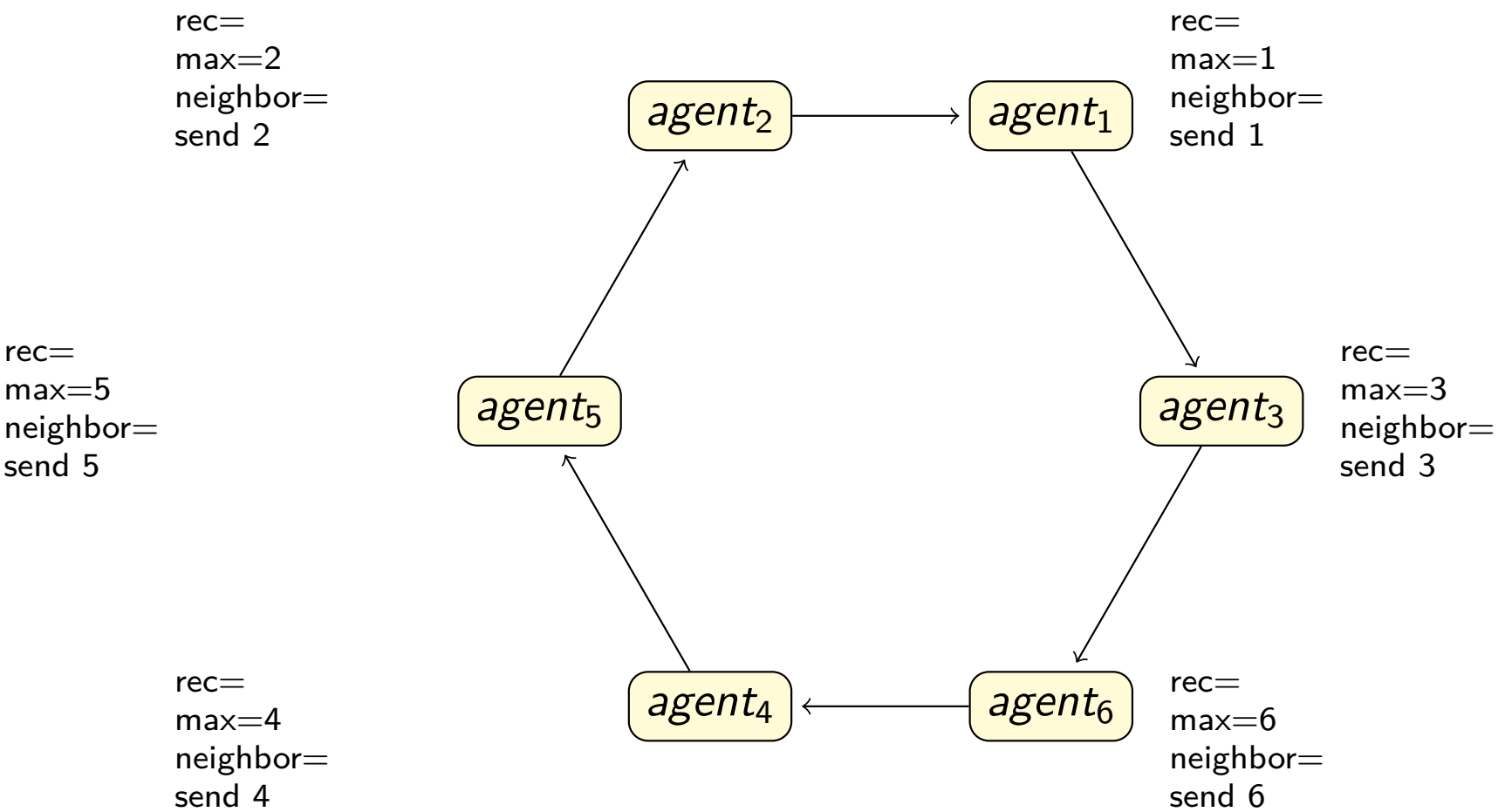
Overview of Promela

- 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

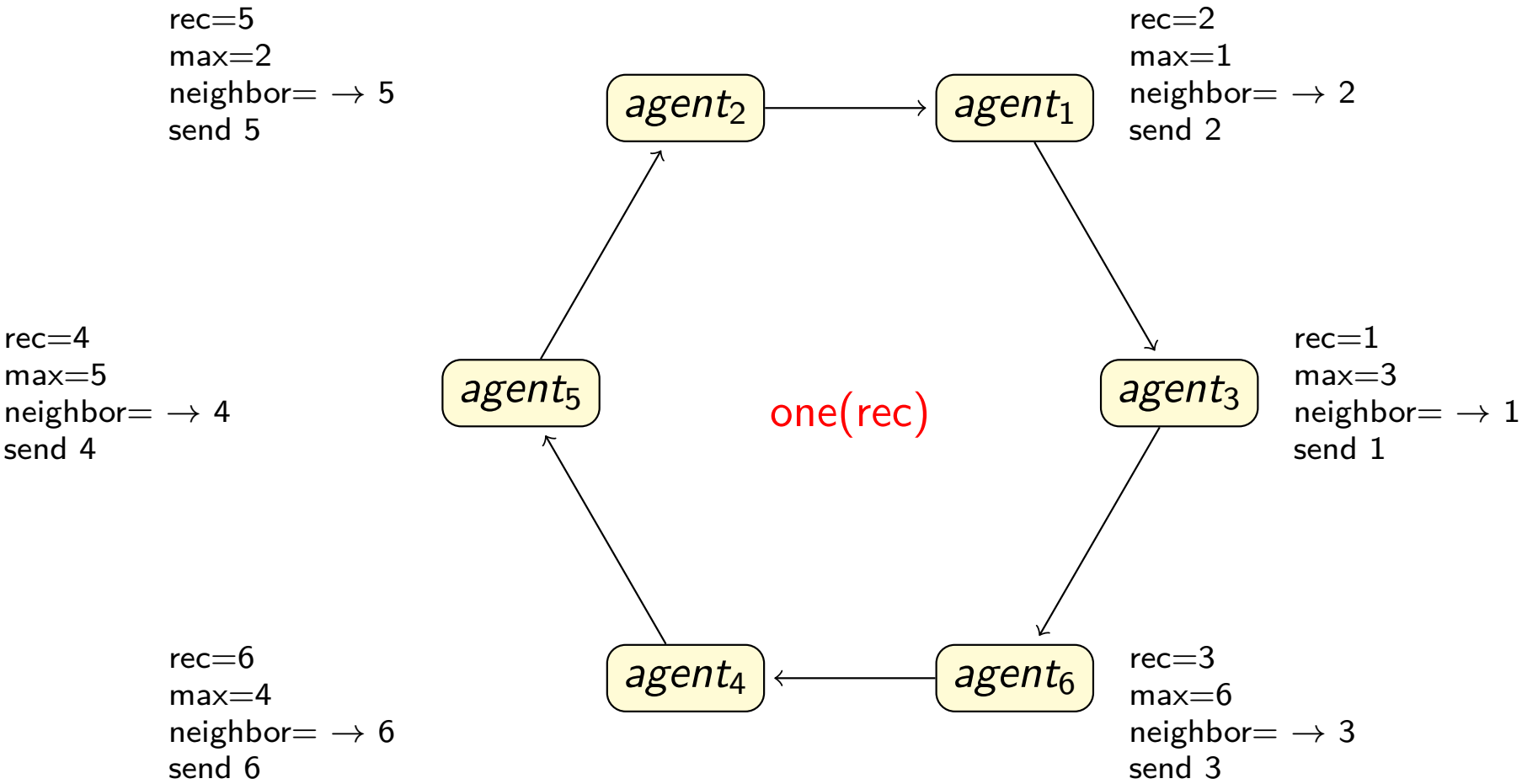
Only finite data domains

Bound on maximal number of concurrent processes

Example: Leader Election

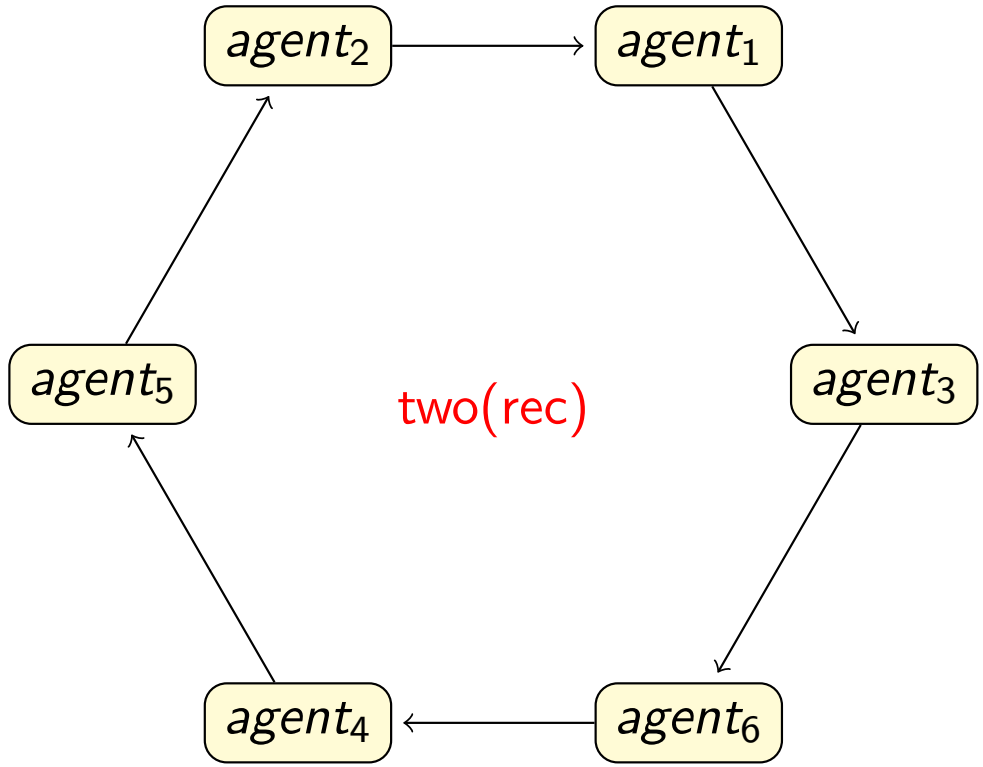


Example: Leader Election



Example: Leader Election

rec=4
max= 2→5
neighbor=5
send 5

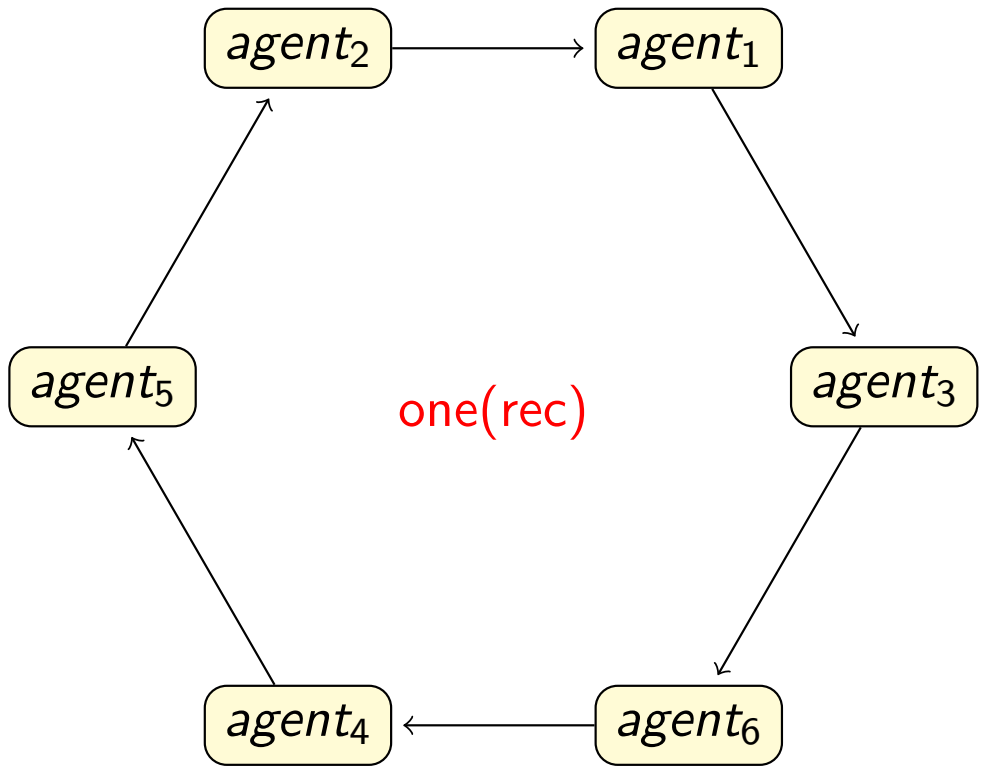


rec=3
max=4→6
neighbor=6
send 6

```
(neighbor > rec && neighbor > max) →
max = neighbor; send one(neighbor)
```

Example: Leader Election

rec=6
max=5
neighbor=5→6
send 6

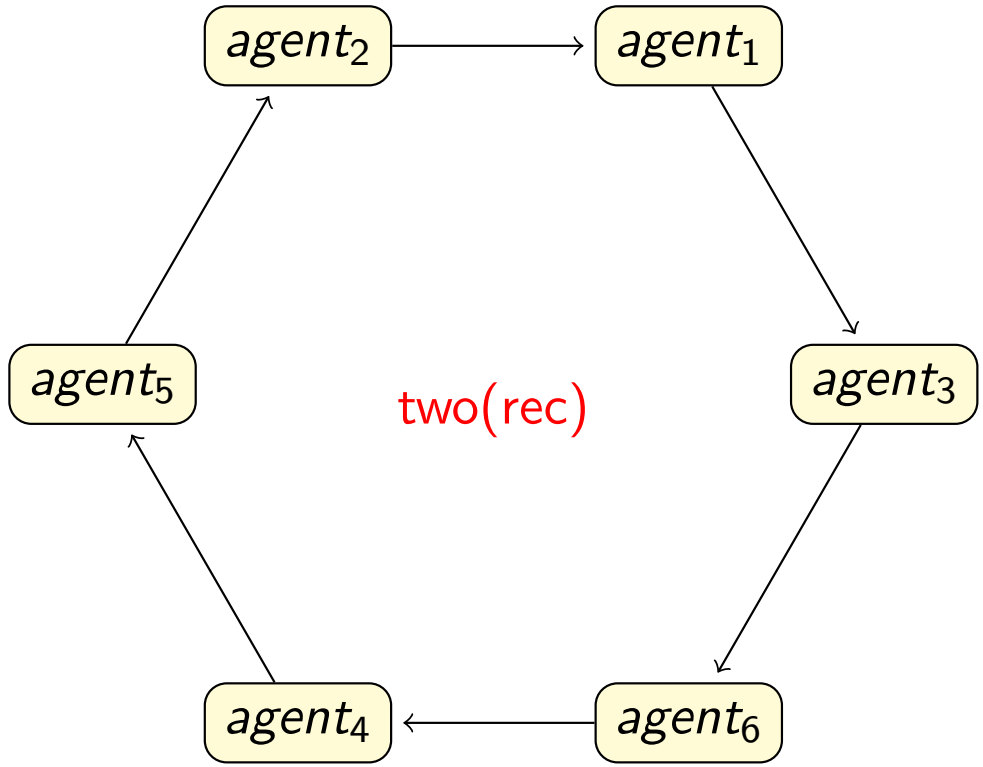


rec=5
max=6
neighbor=6→5
send 5

```
max != rec → neighbor = rec, send two(rec)
```

Example: Leader Election

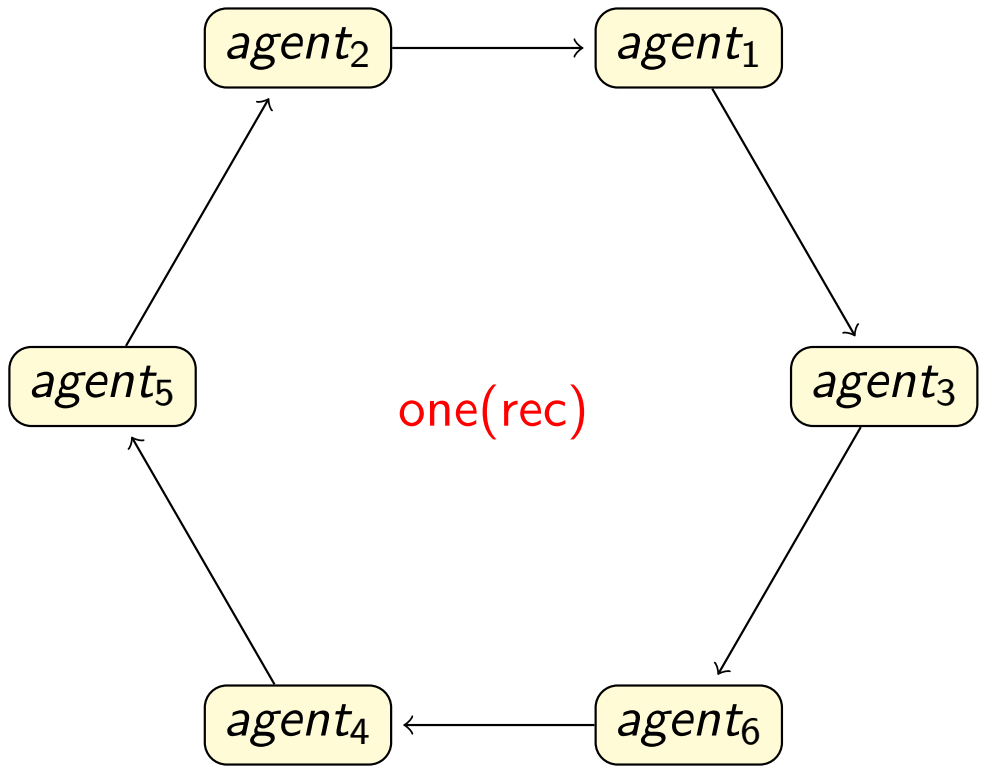
rec=5
max=5→6
neighbor=6
send 6




```
(neighbor > rec && neighbor > max) →
max = neighbor; send one(neighbor)
```

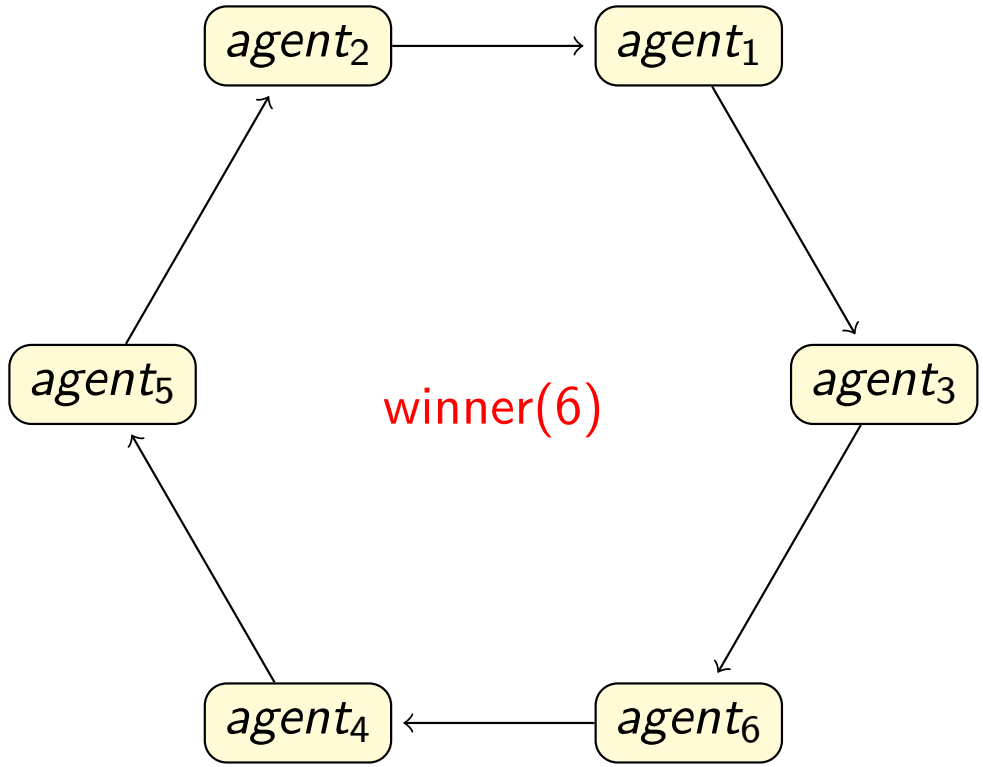
Example: Leader Election

rec=6
max=6
neighbor=6
send 6



`max == rec` $\rightarrow$ `know_winner; send winner(rec)`

Example: Leader Election



`know_winner` $\rightarrow$ `break`

```
#define N 5       /* number 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 cin, cout; byte mynumber) {
   bit Active = 1, know_winner = 0;
   byte rec, maximum = mynumber, neighbor;
   printf("MSC: %d\n", mynumber);
   cout!one(mynumber);
end:    do
   :: cin?one(rec) ->
      if
      :: Active ->
         if
         :: rec != maximum ->
            cout!two(rec);
            neighbor = rec
         :: else ->
            assert(rec == N); /* max is greatest number */
            know_winner = 1;
            cout!winner(rec);
         fi
      :: else ->
         cout!one(rec)
      fi
   :: cin?two(rec) ->
      if
      :: Active ->
         if
         :: neighbor > rec && neighbor > maximum ->
            maximum = neighbor;
            cout!one(neighbor)
         :: else ->
            Active = 0
         fi
      :: else ->
         cout!two(rec)
      fi
   :: cin?winner(rec) ->
      if
      :: rec != mynumber ->
         printf("MSC: LOST\n");
      :: else ->
         printf("MSC: LEADER\n");
         nr_leaders++;
         assert(nr_leaders == 1)
      fi ;
      if
      :: know_winner
      :: else -> cout!winner(rec)
      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       /* number 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 cin, cout; byte mynumber) {
   bit Active = 1, know_winner = 0;
   byte rec, maximum = mynumber, neighbor;
   printf("MSC: %d\n", mynumber);
   cout!one(mynumber);
end:    do
   :: cin?one(rec) ->
      if
      :: Active ->
         if
         :: rec != maximum ->
            cout!two(rec);
            neighbor = rec
         :: else ->
            assert(rec == N); /* max is greatest number */
            know_winner = 1;
            cout!winner(rec);
         fi
      :: else ->
         cout!one(rec)
      fi
   :: cin?two(rec) ->
      if
      :: Active ->
         if
         :: neighbor > rec && neighbor > maximum ->
            maximum = neighbor;
            cout!one(neighbor)
         :: else ->
            Active = 0
         fi
      :: else ->
         cout!two(rec)
      fi
   :: cin?winner(rec) ->
      if
      :: rec != mynumber ->
         printf("MSC: LOST\n");
      :: else ->
         printf("MSC: LEADER\n");
         nr_leaders++;
         assert(nr_leaders == 1)
      fi ;
      if
      :: know_winner
      :: else -> cout!winner(rec)
      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         /* number 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       /* number 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 cin, cout; byte mynumber) {
   bit Active = 1, know_winner = 0;
   byte rec, maximum = mynumber, neighbor;
   printf("MSC: %d\n", mynumber);
   cout!one(mynumber);
end:    do
   :: cin?one(rec) ->
      if
      :: Active ->
         if
         :: rec != maximum ->
            cout!two(rec);
            neighbor = rec
         :: else ->
            assert(rec == N); /* max is greatest number */
            know_winner = 1;
            cout!winner(rec);
         fi
      :: else ->
         cout!one(rec)
      fi
   :: cin?two(rec) ->
      if
      :: Active ->
         if
         :: neighbor > rec && neighbor > maximum ->
            maximum = neighbor;
            cout!one(neighbor)
         :: else ->
            Active = 0
         fi
      :: else ->
         cout!two(rec)
      fi
   :: cin?winner(rec) ->
      if
      :: rec != mynumber ->
         printf("MSC: LOST\n");
      :: else ->
         printf("MSC: LEADER\n");
         nr_leaders++;
         assert(nr_leaders == 1)
      fi ;
      if
      :: know_winner
      :: else -> cout!winner(rec)
      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 cin, cout; byte mynumber) {
   bit Active = 1, know_winner = 0;
   byte rec, maximum = mynumber, neighbor;
   printf("MSC: %d\n", mynumber);
   cout!one(mynumber);
end:
   do
   :: cin?one(rec) -> skip;
   :: cin?two(rec) -> skip;
   od
```

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

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

A receive statement blocks if the match fails

```
#define N 5       /* number 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 cin, cout; byte mynumber) {
   bit Active = 1, know_winner = 0;
   byte rec, maximum = mynumber, neighbor;
   printf("MSC: %d\n", mynumber);
   cout!one(mynumber);
end:    do
   :: cin?one(rec) ->
      if
      :: Active ->
         if
         :: rec != maximum ->
            cout!two(rec);
            neighbor = rec
         :: else ->
            assert(rec == N); /* max is greatest number */
            know_winner = 1;
            cout!winner(rec);
         fi
      :: else ->
         cout!one(rec)
      fi
   :: cin?two(rec) ->
      if
      :: Active ->
         if
         :: neighbor > rec && neighbor > maximum ->
            maximum = neighbor;
            cout!one(neighbor)
         :: else ->
            Active = 0
         fi
      :: else ->
         cout!two(rec)
      fi
   :: cin?winner(rec) ->
      if
      :: rec != mynumber ->
         printf("MSC: LOST\n");
      :: else ->
         printf("MSC: LEADER\n");
         nr_leaders++;
         assert(nr_leaders == 1)
      fi ;
      if
      :: know_winner
      :: else -> cout!winner(rec)
      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

```
:: cin?one(rec) ->
   if
   :: Active ->
      if
      :: rec != maximum ->
         cout!two(rec);
         neighbor = rec
      :: else ->
         assert(rec == N); /* max is greatest number */
         know_winner = 1;
         cout!winner(rec);
      fi
   :: else ->
      cout!one(rec)
   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       /* number 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 cin, cout; byte mynumber) {
   bit Active = 1, know_winner = 0;
   byte rec, maximum = mynumber, neighbor;
   printf("MSC: %d\n", mynumber);
   cout!one(mynumber);
end:    do
   :: cin?one(rec) ->
      if
      :: Active ->
         if
         :: rec != maximum ->
            cout!two(rec);
            neighbor = rec
         :: else ->
            assert(rec == N); /* max is greatest number */
            know_winner = 1;
            cout!winner(rec);
         fi
      :: else ->
         cout!one(rec)
      fi
   :: cin?two(rec) ->
      if
      :: Active ->
         if
         :: neighbor > rec && neighbor > maximum ->
            maximum = neighbor;
            cout!one(neighbor)
         :: else ->
            Active = 0
         fi
      :: else ->
         cout!two(rec)
      fi
   :: cin?winner(rec) ->
      if
      :: rec != mynumber ->
         printf("MSC: LOST\n");
      :: else ->
         printf("MSC: LEADER\n");
         nr_leaders++;
         assert(nr_leaders == 1)
      fi ;
      if
      :: know_winner
      :: else -> cout!winner(rec)
      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
   }
}
```

```
:: cin?two(rec) ->
   if
   :: Active ->
      if
      :: neighbor > rec && neighbor > maximum ->
         maximum = neighbor;
         cout!one(neighbor)
      :: else ->
         Active = 0
      fi
   :: else ->
      cout!two(rec)
   fi
```

```
#define N 5       /* number 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 cin, cout; byte mynumber) {
   bit Active = 1, know_winner = 0;
   byte rec, maximum = mynumber, neighbor;
   printf("MSC: %d\n", mynumber);
   cout!one(mynumber);
end:    do
   :: cin?one(rec) ->
      if
      :: Active ->
         if
         :: rec != maximum ->
            cout!two(rec);
            neighbor = rec
         :: else ->
            assert(rec == N); /* max is greatest number */
            know_winner = 1;
            cout!winner(rec);
         fi
      :: else ->
         cout!one(rec)
      fi
   :: cin?two(rec) ->
      if
      :: Active ->
         if
         :: neighbor > rec && neighbor > maximum ->
            maximum = neighbor;
            cout!one(neighbor)
         :: else ->
            Active = 0
         fi
      :: else ->
         cout!two(rec)
      fi
   :: cin?winner(rec) ->
      if
      :: rec != mynumber ->
         printf("MSC: LOST\n");
      :: else ->
         printf("MSC: LEADER\n");
         nr_leaders++;
         assert(nr_leaders == 1)
      fi ;
      if
      :: know_winner
      :: else -> cout!winner(rec)
      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

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

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

```
#define N 5       /* number 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 cin, cout; byte mynumber) {
   bit Active = 1, know_winner = 0;
   byte rec, maximum = mynumber, neighbor;
   printf("MSC: %d\n", mynumber);
   cout!one(mynumber);
end:    do
   :: cin?one(rec) ->
      if
      :: Active ->
         if
         :: rec != maximum ->
            cout!two(rec);
            neighbor = rec
         :: else ->
            assert(rec == N); /* max is greatest number */
            know_winner = 1;
            out!winner(rec);
         fi
      :: else ->
         cout!one(rec)
      fi
   :: cin?two(rec) ->
      if
      :: Active ->
         if
         :: neighbor > rec && neighbor > maximum ->
            maximum = neighbor;
            cout!one(neighbor)
         :: else ->
            Active = 0
         fi
      :: else ->
         cout!two(rec)
      fi
   :: cin?winner(rec) ->
      if
      :: rec != mynumber ->
         printf("MSC: LOST\n");
      :: else ->
         printf("MSC: LEADER\n");
         nr_leaders++;
         assert(nr_leaders == 1)
      fi ;
      if
      :: know_winner
      :: else -> cout!winner(rec)
      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

```
      :: cin?winner(rec) ->
         if
         :: rec != mynumber ->
            printf("MSC: LOST\n");
         :: else ->
            printf("MSC: LEADER\n");
            nr_leaders++;
            assert(nr_leaders == 1)
         fi ;
         if
         :: know_winner
         :: else -> cout!winner(rec)
         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