Verifying a distributed leader election algorithm using the Spin Model Checker

João Paulo Pizani Flor
j.p.pizaniflor@students.uu.nl

Tuesday 26th February, 2013

1 Introduction

This technical report describes the solution adopted to one of the programming assignments of the master course "Program Verification" at Utrecht University, taught in the 3rd period of the academic year 2012/2013. The assignment involved the modelling of a distributed algorithm achieving leader election and the formal verification of said algorithm using model checking techniques.

More specifically, the "Chang and Roberts" decentralized leader election algorithm was modeled, with the following characteristics:

- N processes are connected in a directed ring topology
- Each process has exactly one incoming and one outgoing channel

Two correctness properties over the model were then verified:

1. The algorithm always terminates
2. When the algorithm is finished, all processes agree on who is the new leader

These properties were expressed using Linear-time Temporal Logic (LTL) formulas, which were negated and translated to *never claims* verified by the SPIN model checker.

2 The Promela model

In order to verify the correctness properties proposed, a model of the algorithm was written in the Process Metalanguage (Promela). The developed model has a compile-time constant (NPROC) defining the number of processes in the ring. As model checking techniques can only handle finite state spaces, we cannot do universal quantification over the number of processes, and only rings up to a certain size were verified. More details about verification results are in section 4

Besides the variables modelling the algorthm itself, the Promela file that describes the algorithm (verkiezingen.pml) also contains 3 so called *shadow variables*:

```
byte terminating_procs = 0;
byte proclaimed_leader = 0;
byte disagreement_leader = 0;
```

The first variable (`terminating_procs`) is increased whenever a process reaches the default end state (last line of its body). Therefore, the value of this variable serves as a count of how many of the instantiated processes actually terminated.

The two following variables (`proclaimed_leader` and `disagreement_leader`) are used to establish the property that all processes agree on the leader.

- The winner of the election, when first noticing his victory, writes his id to the global shadow variable `proclaimed_leader`.
- Whenever a process assigns its local `lid` (leader) variable to an id, it also updates the shadow `disagreement_leader` to denote whether there are any differences between its idea of a leader and the global proclaimed leader.

The Promela code for the init section follows, which contains an array serving as a "source" of unique ids to the processes being created, as well as the loop itself which runs the processes.

```
init {
    byte i = 0;
    byte ids[20];

    atomic {
        ids[0] = 32; ids[1] = 45; ids[2] = 12; ids[3] = 21;
        ids[4] = 10; ids[5] = 11; ids[6] = 99; ids[7] = 87;
        ids[8] = 41; ids[9] = 37; ids[10] = 31; ids[11] = 56;
        ids[12] = 78; ids[13] = 26; ids[14] = 17; ids[15] = 23;
        ids[16] = 42; ids[17] = 22; ids[18] = 55; ids[19] = 75;
    }

    atomic {
        do
        :: i < NPROC -> run P(c[i], c[(i+1) % NPROC], ids[i]); i++;
        :: else      -> break;
        od;
    }
}
```

The following code block shows the declarations of the channels used for inter-process communication, as well as the message type.

```
#define NPROC 15

mtype = { ELECTION, NEW_LEADER };
chan c[NPROC] = [1] of { mtype, byte };
```

The fact that two kinds of messages can be sent across the channels is justified by existance of two *phases* in the election algorithm:

Election The processes cooperate to choose one of them as the new leader

Acknowledgement All processes recognize the new leader

In the election phase, messages with the tag `ELECTION` are exchanges. And in the second phase of the algorithm, only messages with the tag `NEW_LEADER` are passed around. The following code block is the proctype declaration of the member processes.

```
proctype P(chan i; chan o; byte id) {
    mtype t;
    byte buf;
    byte lid;
    printf("my pid is: %d\n", id);

    o ! ELECTION, id;
    do
    :: {
        i ? t, buf;
        if
        :: (t == ELECTION) -> if
            :: (buf > id) -> printf("(%d>%d)\n", buf, id);
                             o ! ELECTION, buf;
            :: (buf < id) -> printf("(%d<%d)\n", buf, id);
            :: else       -> {
                    lid = id;
                    proclaimed_leader = id;  // shadow var
                    o ! NEW_LEADER, lid;
                }
            fi;
        :: (t == NEW_LEADER) -> if
            :: (buf == id) -> {
                    printf("election ended, winner is %d\n", id);
                    break;
                }
            :: else -> {
                    lid = buf;
                    // shadow var
                    disagreement_leader = disagreement_leader + (proclaimed_leader -
                    printf("ACK leader is %d\n", lid);
                    o ! NEW_LEADER, lid; break;
                }
            fi;
        fi;
       }
    od;
einde: terminating_procs++;  // shadow var, this process terminates
}
```

As visible above, each process has two variables of type `byte` related to the election algorithm: one (`buf`) where it stores the contents received over the incoming channel, and the other (`lid`) which stores the id of the process believed to be the leader.

2.1 Non-deterministic election

There is also a non-deterministic version of the election algorithm, which model is in the file `verkiezingen_nd.pml`. This model works through a change in the second phase of the election process (acknowledgment). Whenever a process gets a `NEW_LEADER` message, it has a (non-deterministic) choice between accepting that proposed leader or attempting a *coup d'état*, that is, telling his neighbour that HE is the leader, instead of the "legitimate" one. This possibility of coups can only last for a limited number of rounds (limited by MAXCOUPS).

This non-deterministic version has also been verified, and both LTL formulas were checked valid with SPIN.

3 Correctness properties

As already mentioned, the assignment asked for the verification of two correctness properties over the described model:

1. The algorithm always terminates
2. When the algorithm is finished, all processes agree on who is the new leader

The verification was realized by first expressing each of the desired properties as a formula in Linear-time Temporal Logic (LTL). The first formula, which expresses the fact that all processes terminate, is as follows:

$$\Box \diamond (t = N)$$

That is, it is *always* the case that *eventually* a state will be reached in which $t = N$, where t is the `terminating_procs` shadow variable and N is `NPROC`.

The second formula expresses the fact that when the algorithm finishes, all processes agree on who is the new leader:

$$\Box (t = N \implies d = 0)$$

That is, it is *always* the case that when the number of terminating process matches the total of all processes, the "accumulated" disagreement will be zero, where t is `terminating_procs`, N is `NPROC` and d is `disagreement_leader`.

Even though this task has not been finished, we believe that it would be possible to express the second property (*agreement*) without the use of LTL, only using a monitor process with an assertion. The first property, however, could not be expressed without the use of LTL.

4 Results

In this section the verification results are shown. Three scenarios were considered for verification, regarding the number of processes in the ring: we tested the algorithm with 5, 10 and 15 processes.

In the case of 15 processes, the number of states to be explored was already to big for the verification to run on my machine (4GB of RAM), therefore the verification was not completed, but achieved a depth of around 800.

Here are the verification results for the case with 5 processes:

```
# VERIFYING TERMINATION
(Spin Version 6.2.3 -- 24 October 2012)
        + Partial Order Reduction

Full statespace search for:
        never claim             + (termination)
        assertion violations    + (if within scope of claim)
        cycle checks            - (disabled by -DSAFETY)
        invalid end states      - (disabled by never claim)

State-vector 128 byte, depth reached 204, errors: 0
     2095 states, stored
```

```
     2415 states, matched
     4510 transitions (= stored+matched)
       51 atomic steps
hash conflicts:         3 (resolved)

Stats on memory usage (in Megabytes):
    0.280       equivalent memory usage for states (stored*(State-vector + overhead))
    0.480       actual memory usage for states
   64.000       memory used for hash table (-w24)
    0.343       memory used for DFS stack (-m10000)
   64.734       total actual memory usage

pan: elapsed time 0.01 seconds

# VERIFYING AGREEMENT
(Spin Version 6.2.3 -- 24 October 2012)
        + Partial Order Reduction

Full statespace search for:
        never claim             + (agreement)
        assertion violations    + (if within scope of claim)
        cycle checks            - (disabled by -DSAFETY)
        invalid end states      - (disabled by never claim)

State-vector 128 byte, depth reached 204, errors: 0
     1049 states, stored
      460 states, matched
     1509 transitions (= stored+matched)
       17 atomic steps
hash conflicts:         0 (resolved)

Stats on memory usage (in Megabytes):
    0.140       equivalent memory usage for states (stored*(State-vector + overhead))
    0.383       actual memory usage for states
   64.000       memory used for hash table (-w24)
    0.343       memory used for DFS stack (-m10000)
   64.636       total actual memory usage

pan: elapsed time 0 seconds
```

Now, for the case where there are 10 processes in the ring, the number of states goes from one thousand to around one million.

```
# VERIFYING TERMINATION
(Spin Version 6.2.3 -- 24 October 2012)
        + Partial Order Reduction

Full statespace search for:
        never claim             + (termination)
        assertion violations    + (if within scope of claim)
        cycle checks            - (disabled by -DSAFETY)
        invalid end states      - (disabled by never claim)

State-vector 212 byte, depth reached 527, errors: 0
  1111711 states, stored
```

```
  1619546 states, matched
  2731257 transitions (= stored+matched)
       96 atomic steps
hash conflicts:     35590 (resolved)

Stats on memory usage (in Megabytes):
  237.487       equivalent memory usage for states (stored*(State-vector + overhead))
  191.500       actual memory usage for states (compression: 80.64%)
                state-vector as stored = 169 byte + 12 byte overhead
   64.000       memory used for hash table (-w24)
    0.343       memory used for DFS stack (-m10000)
  255.457       total actual memory usage

pan: elapsed time 4.77 seconds
pan: rate  233063.1 states/second


# VERIFYING AGREEMENT
(Spin Version 6.2.3 -- 24 October 2012)
        + Partial Order Reduction

Full statespace search for:
        never claim             + (agreement)
        assertion violations    + (if within scope of claim)
        cycle checks            - (disabled by -DSAFETY)
        invalid end states      - (disabled by never claim)

State-vector 212 byte, depth reached 527, errors: 0
   555857 states, stored
   354571 states, matched
   910428 transitions (= stored+matched)
       32 atomic steps
hash conflicts:      3508 (resolved)

Stats on memory usage (in Megabytes):
  118.744       equivalent memory usage for states (stored*(State-vector + overhead))
   95.941       actual memory usage for states (compression: 80.80%)
                state-vector as stored = 169 byte + 12 byte overhead
   64.000       memory used for hash table (-w24)
    0.343       memory used for DFS stack (-m10000)
  160.047       total actual memory usage

pan: elapsed time 1.6 seconds
pan: rate 347410.62 states/second
```

And finally the largest test case, where we had 15 processes in the ring. In this case, as already mentioned, the verification could not be done exhaustively become of memory limitations of the computer in which the verification was run. The depth we could reach was about 800 and the number of states we managed to verify was approximately 25 million.

```
# VERIFYING TERMINATION
Warning: Search not completed
        + Partial Order Reduction
        + Compression
```

```
Full statespace search for:
        never claim             + (termination)
        assertion violations    + (if within scope of claim)
        cycle checks            - (disabled by -DSAFETY)
        invalid end states      - (disabled by never claim)

State-vector 296 byte, depth reached 829, errors: 0
 25352927 states, stored
 16224078 states, matched
 41577005 transitions (= stored+matched)
       47 atomic steps
hash conflicts:  10589918 (resolved)

Stats on memory usage (in Megabytes):
 7543.672       equivalent memory usage for states (stored*(State-vector + overhead))
 1373.926       actual memory usage for states (compression: 18.21%)
                state-vector as stored = 41 byte + 16 byte overhead
   64.000       memory used for hash table (-w24)
    0.343       memory used for DFS stack (-m10000)
 1437.781       total actual memory usage

nr of templates: [ 0:globals 1:chans 2:procs ]
collapse counts: [ 0:199249 2:3 3:390 4:2 ]

pan: elapsed time 172 seconds
pan: rate  147615.3 states/second

# VERIFYING AGREEMENT
(Spin Version 6.2.3 -- 24 October 2012)
Warning: Search not completed
        + Partial Order Reduction
        + Compression

Full statespace search for:
        never claim             + (agreement)
        assertion violations    + (if within scope of claim)
        cycle checks            - (disabled by -DSAFETY)
        invalid end states      - (disabled by never claim)

State-vector 296 byte, depth reached 834, errors: 0
 30224192 states, stored
 19264186 states, matched
 49488378 transitions (= stored+matched)
       47 atomic steps
hash conflicts:  15757143 (resolved)

Stats on memory usage (in Megabytes):
 8993.099       equivalent memory usage for states (stored*(State-vector + overhead))
 1636.211       actual memory usage for states (compression: 18.19%)
                state-vector as stored = 41 byte + 16 byte overhead
   64.000       memory used for hash table (-w24)
    0.343       memory used for DFS stack (-m10000)
 1699.988       total actual memory usage
```

```
nr of templates: [ 0:globals 1:chans 2:procs ]
collapse counts: [ 0:222974 2:3 3:390 5:1 ]

pan: elapsed time 203 seconds
pan: rate 148961.03 states/second
```