Technical University of Denmark
DTU Compute

Concurrent programming
Fall 2014

Car control

Due November 19, 2014

Authors: Group 24

Michael Bøndergaard
(s113112)

Kristoffer Breitenstein
(s113135)

Contents

1 Introduction

The course teaches how to handle software with additional tasks within one software program. Software will have threads handle different tasks and have shared variables, which the threads will have limited or full access to. Concurrent thread can be used for many different objects and goals depending on the kind of project the software is written for. Threads can often be used as optimization in software, where different threads will handle different part of an calculation or another task. Especially with the modern processors running multiple cores at once, then one core can handle a task while another handles a different task.

The project is about concurrent processes running individually and using specific data in cooperation with the other processes. The project has 9 cars driving around a parking lot with the cars having different routes. The cars will have to make sure they do not cause accidents or end up in a dead lock situation. The cars will have to pass through an alley with only room the cars driving in one direction. The cars are run by a thread each to make sure the cars are not getting into an accident.
In the project the alley is managed by either a semaphore or a monitor written in Java. These are two different approaches of how to handle atomic actions in software. In other words mean the variable or a critical section is only available to one thread at a time.

In the end the cars will be driving with the alley acting as a traffic light. The cars only being able to drive in one direction at a time, while the others will have to wait till the alley is available. The cars also has a barrier, where the cars will wait until a specific number of cars are waiting. The traffic will end up going around the circuit in a smooth flow.

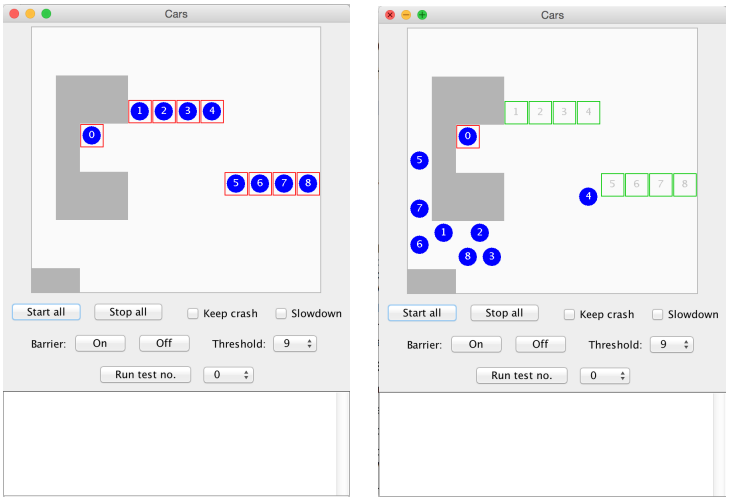
2 Step 1

The system given has the cars driving around the circuit in their specific routes. The cars does not stop, when they bump into each other, instead a red square is shown for each collision.

Step one's requirements are to

- make sure the cars do not collide
- cause deadlocks in the traffic
- have cars drive through the alley

Figure 1: The cars driving around the parking lot



3 Step 2

The software spin uses Promela models to verify a concurrent model. The model shows rather the processes uses variables and other shared features in a valid and intended way. The processes should not be allowed to enter more than 4 cars at a time and only in one direction.

3.1 Analysis

The Promela code is a little different to get to work than the java code. The semaphore described previously result in the code found in appendix A. The code display the enter part followed by the leave part of the cars alley class in java. The result of the analysis shows with 8 threads the program runs correctly.

```
pan: resizing hashtable to -w29.. pan: out of memory
hint: to reduce memory, recompile with
  -DCOLLAPSE # good, fast compression, or
```

```
  -DMA=84   # better/slower compression, or
  -DHC # hash-compaction, approximation
  -DBITSTATE # supertrace, approximation
(Spin Version 6.1.0 -- 4 May 2011)
Warning: Search not completed
        + Partial Order Reduction
Full statespace search for:
        never claim             - (none specified)
        assertion violations    +
        cycle checks            - (disabled by -DSAFETY)
        invalid end states      +
State-vector 84 byte, depth reached 108, ... errors: 0 ...
 2.69e+08 states, stored
6.5625499e+08 states, matched
9.2525499e+08 transitions (= stored+matched)
1.4695379e+08 atomic steps
hash conflicts: 4.0118364e+08 (resolved)
Stats on memory usage (in Megabytes):
28732.300       equivalent memory usage for
states (stored*(State-vector + overhead))
11478.569       actual memory usage for states (compression: 39.95%)
                state-vector as stored = 17 byte + 28 byte overhead
 4096.000       memory used for hash table (-w29)
    0.107       memory used for DFS stack (-m2000)
    4.417       memory lost to fragmentation
15570.259       total actual memory usage
pan: elapsed time 1.49e+03 seconds
pan: rate     180388 states/second
```

The analysis does not show any errors, but a warning is given "Search not complete". Other analysis have been run, which does not show the warning, but these are with less processes. These analysis can be found in the appendix A. The problem is that the pan runs out of memory on the thin clients, which result in a termination. The program does the analysis, but takes a long time and so far the biggest analysis run has 6 processes without error. The error can occur on 6 processes, but does not occur every time. For 5 processes the analysis does seem to work every time, but does only have 1 car in the one direction and 4 in the other.
The analysis shows no errors which means the checks (found at the bottom of the code) do hold. The different semaphores are kept within the limits of the variables.

3.2 Implementation

The code in appendix A is a close implementation of the code in java, which can be found in appendix B

4 Step 3

5 Step 4

6 Step 5

7 Tests

8 Evaluation

9 Conclusion

10 Appendix

A jSpin

The code from jSpin to run the analysis:

```
define N                    8       /* no. of processes */

short c = 0;
short u = 4;
short d = 1;
short b = 1;
bool trafficUp = false;

/* Declare and instantiate N Counter processes */


inline v(s){
        s++;
}

inline p(s){
        atomic{
                s > 0 -> s--;
        }
}

active [N] proctype Alley ()
{
        c=(c+1)%N;
entry:
        if :: c<4 ->
                if :: trafficUp == true ->
                        if :: u == 4 -> p(d);
                                trafficUp = true;
                :: else -> skip;
                        fi;
```

```
                p(u);
                :: trafficUp == false -> p(b);
                        if :: trafficUp == true ->
                                v(b);
                        :: trafficUp == false ->
                                p(d);
                                v(b);
                        fi;
                        p(u);
                        trafficUp=true;
                :: else -> skip;
                fi;
        :: c>=4 ->
                if :: trafficUp == true ->
                        p(d);
                        p(u);
                        trafficUp = false;
                :: trafficUp ==false; ->
                        if :: u == 4 ->
                                p(d);
                                trafficUp=false;
                        :: else -> skip;
                        fi;
                        p(u);
                fi;
        :: else -> skip;
        fi;

leave:
        v(u);
        atomic{
        if :: u==4 ->
                if :: d== 0 -> v(d);
                :: else -> skip;
                fi;
        :: else -> skip;
        fi;
        }
}

/* Invariant check */
active proctype Check ()
{
        (0 <= c && c <= 7) -> assert(true);
        (0 <= u && u <= 4) -> assert(true);
        (0 <= d && d <= 1) -> assert(true);
        (0 <= b && b <= 1) -> assert(true);
}
```

The analysis run with 5 processes:

```
(Spin Version 6.1.0 -- 4 May 2011)
        + Partial Order Reduction
Full statespace search for:
        never claim             - (none specified)
        assertion violations    +
        cycle checks            - (disabled by -DSAFETY)
        invalid end states      +
State-vector 60 byte, depth reached 72, ... errors: 0 ...
 36981340 states, stored
 57403474 states, matched
 94384814 transitions (= stored+matched)
 10018950 atomic steps
hash conflicts:  35003377 (resolved)
Stats on memory usage (in Megabytes):
 3103.598       equivalent memory usage for states
 (stored*(State-vector + overhead))
 2015.764       actual memory usage for states (compression: 64.95%)
                state-vector as stored = 29 byte + 28 byte overhead
  256.000       memory used for hash table (-w25)
    0.107       memory used for DFS stack (-m2000)
 2271.064       total actual memory usage
unreached in proctype Alley
        (0 of 91 states)
unreached in proctype Check
        (0 of 9 states)
pan: elapsed time 145 seconds
pan: rate 255784.62 states/second
```

The analysis run with 6 process:

```
pan: resizing hashtable to -w29.. pan: out of memory
hint: to reduce memory, recompile with
  -DCOLLAPSE # good, fast compression, or
  -DMA=68    # better/slower compression, or
  -DHC # hash-compaction, approximation
  -DBITSTATE # supertrace, approximation
(Spin Version 6.1.0 -- 4 May 2011)
Warning: Search not completed
        + Partial Order Reduction
Full statespace search for:
        never claim             - (none specified)
        assertion violations    +
        cycle checks            - (disabled by -DSAFETY)
        invalid end states      +
State-vector 68 byte, depth reached 84, ... errors: 0 ...
 2.69e+08 states, stored
4.5920728e+08 states, matched
7.2820728e+08 transitions (= stored+matched)
 79882859 atomic steps
hash conflicts: 2.7405665e+08 (resolved)
Stats on memory usage (in Megabytes):
```

```
24627.686       equivalent memory usage for
states (stored*(State-vector + overhead))
11490.506       actual memory usage for states (compression: 46.66%)
                state-vector as stored = 17 byte + 28 byte overhead
 4096.000       memory used for hash table (-w29)
    0.107       memory used for DFS stack (-m2000)
    4.537       memory lost to fragmentation
15582.076       total actual memory usage
pan: elapsed time 1.16e+03 seconds
pan: rate 231499.41 states/second
```

B Java