Technische Universität München **Summer Semester 2015**
I7
J. Esparza / A. Durand-Gasselin / D. Suwimonteerabuth **21.05.2015**

Model Checking – Exercise sheet 6

Exercise 6.1

Lamport's bakery algorithm implements mutual exclusion by drawing an analogy between customers waiting in a bakery and processes waiting to access a critical section. The idea is to make sure that one customer can be served at a time by means of a numbering machine. Upon entering the bakery, each customer receives a unique number, and waits for the number's turn. The machine increases the number by one for each customer. There is a global counter in the bakery which displays the number of customer that is currently being served. When the customer leaves, the counter is increased by one to indicate the next customer's turn.

Read the original paper for more details: `http://research.microsoft.com/en-us/um/people/lamport/pubs/bakery.pdf`

1. Assuming that there are only two processes, write down a Promela model for the algorithm.
2. Write down LTL formulae for the following properties:
   (a) Both processes cannot enter the critical section at the same time.
   (b) After a process receives a number, it will eventually enter the critical section.
3. Use Spin to verify your model against the above properties.

Exercise 6.2

Consider below an implementation of Insertion sort

```c
#include <stdio.h>

#define N 10

int main() {
  int a[N];

  int i;
  for (i = 1; i < N; i++) {
    int x = a[i];
    int j = i;
    while (j > 0 && a[j - 1] > x) {
      a[j] = a[j - 1];
      j--;
    }
    a[j] = x;
  }
}
```

1. Write down a Promela model for the above C function

2. Check your model against the following properties:

   (a) The function always terminates.

   (b) When the function terminates, the array is correctly sorted.

   (c) After executing line 16, the array elements between `a[0]` and `a[i]` are sorted.

Exercise 6.3

Consider the following Promela model:

```
1  #define N 1
2
3  active proctype a() {
4    byte x;
5    do
6    :: (x < N) -> x++
7    :: else -> break
8    od;
9    assert(x == N);
10 }
11
12 active proctype b() {
13   byte y;
14   do
15   :: (y < N) -> y++
16   :: else -> break
17   od;
18   assert(y == N);
19 }
```

1. Draw a Kripke structure for the above model.

2. Write down the *independence relation* for the Kripke structure.

3. Use Spin to verify the above model *with* and *without* using partial order reduction. How many states did Spin search for in each case? Explain the difference.

4. Increase `N` to 10, 100, 200, and redo the previous step. Observe the number of states.

5. Modify the model by

   - making `x` a global variable; and
   - appending `acceptb: skip` to the last line of `b`.

   Verify the following property with Spin:

   ```
   ((x == 0) U (x == 1)) && <> (b@acceptb)
   ```

   (a) Compare the number of states *with* and *without* using partial order reduction.

   (b) Which actions are now *visible*?