Spin: Exercises*

Patrick Trentin
patrick.trentin@unitn.it
http://disi.unitn.it/trentin

Formal Methods Lab Class, March 31, 2017

Università degli Studi di Trento

---

*These slides are derived from those by Stefano Tonetta, Alberto Griggio, Silvia Tomasi, Thi Thieu Hoa Le, Alessandra Giordani, Patrick Trentin for FM lab 2005/16

Exercise 1: mutual exclusion [1/2]

Exercise: A solution to **mutual exclusion** for **N processes** is based on message passing instead of shared variables.

Idea: use a shared **buffered** message channel and synchronize by reading and writing from/onto this channel.

- the only shared global data structure can be a channel
- check with **ltl** that following properties hold for 3 processes:
  - mutual exclusion
  - progress
  - lockout-freedom

- **Q:** why is the fairness condition necessary for the **lockout-freedom** property to hold?
- **Q:** What changes if a **synchronous** channel is used instead? (why?)

Exercise 1: mutual exclusion [2/2]

Idea: replace the channel-based synchronization mechanism of **Exercise 1** with the famous **Test and Set** solution:

```
...
// enter critical section
do
  :: atomic {
     tmp = lock;
     lock = true;
     } ->
     if
        :: tmp;
        :: else -> break;
     fi;
od;
...
```

```
// global variable
bool lock = false
...
// exit critical section
lock = false;
...
```

Q: does the program still verify all the properties? (why?)

Exercise 2: factorial

Exercise: Model a process **factorial(n, c)** that **recursively** computes the factorial of a given value "n".

Hints & Tasks:

- use **channel** "c" to return the value to your parent process
- spawn the first factorial() process in the **init** block
- verify that fact(k) is greater than 2^k for $k > 3$. (e.g., try with $k = 10$)

Q:

- does the model always terminate, for any given value?
- if not, could you modify the solution so to be complete, whilst also performing all the computation in a recursive fashion? (why?)

Exercise 3: oscillator

Exercise: Write a Promela model that initializes a global integer variable **sum** to be 0. Model a process **P**, stuck in an infinite loop, which:

- draws a random value included in $\{1, 3\}$ and assigns it to **v**
- updates the value of **sum** as follows:
  - if **sum** is positive valued, it subtracts **v** to its value
  - otherwise, it adds **v** to its value

Verify the following **ltl** properties:

- the value of **sum** is equal to 0 infinitely often
- the value of **sum** is never larger than 3 or smaller than -3
- it always the case that if **sum** is greater than 0 then it will eventually be smaller than 0, and if **sum** is smaller than 0 then it will eventually be larger than 0

Q: why is the third property **not** verified? can you fix it?

Exercise 4: jumping array

Exercise: Model an array of **k** elements with **k-1** (random) memory locations initialized to 0 and **one** (random) location initialized to 1. Write an **algorithm** of your choice that searches the array for the memory location with value 1 and terminates only when it finds it. Each time that your algorithm reads **any** memory location, and before the next read, one of the following things must happen at random:

- the value 1 in location i jumps to location $(i+1)\%k$
- the value 1 in location i jumps to location $(i-1)\%k$
- the value 1 in location i does not move

Verify with **ltl** that the algorithm **always** terminates for **k=11**, use option "-mN" to control the **maximum depth** and "-i" for **breadth first** search.

- **Q:** is it possible to verify the correctness of your algorithm? why?
- **Q:** what is the most efficient algorithm (no cheats) for this problem?

Exercises Solutions

- will be uploaded on course website within a couple of days
- send me an email if you need help or you just want to propose your own solution for a review

- learning programming languages requires practice: try to come up with your own solutions first!