Spin exercises

Enrico Magnago

University of Trento,
Fondazione Bruno Kessler

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:

- what happens if we try to compute the factorial of 100?

Exercise 3: 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?

Exercise 4: infinite monkey theorem

Exercise: Model a system of 26 *monkeys* and one human *reviewer*.

- Each **monkey** is given a button which, when pressed, sends a **unique** lower-case character (in the set a..z) to the reviewer. A monkey can press a button at any time, up until when the experiment is over.
- The **reviewer** checks the incoming sequence of characters, one by one, against a famous quote taken from the *Hamlet*: *"to be or not to be"* (spaces and punctuation marks are ignored). As soon as there is a match, the reviewer terminates the experiment.

Use a global, shared, channel `typewriter` to send characters from the *monkeys* to the human *reviewer*.
Write an *LTL* property s.t. the corresponding counter-example found by spin is an execution trace matching the sequence of characters `tobeornottobe`, and use *Spin* to find it.

Q: how can one guarantee correct termination for all processes?

Exercise 5: Elaaden Vault

Exercise: Five ControlPillars, numbered from 0 to 4, control the gate of an ancient vault. Initially, pillars 1, 3 and 4 are in `ON` state, while 0 and 2 are OFF. The gate opens when all pillars are contemporarily set to `ON`.

- Each **ControlPillar** waits for input commands sent through their input channel `ctl`. Whenever a pillar receives a command, it **atomically** changes its own state –and the state of its immediate left and right neighbours– to the opposite value. To this extent, pillars 0 and 4 must be considered neighbours of each other.
- A spaceship **Commander** keeps sending command messages to randomly chosen control pillars, up until the gate opens.

Write a property `p1` s.t. its counter-example is a sequence of button-switches that will open the gate.

Exercise 6: 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 7: cigarette smokers

Exercise: Assume that a cigarette requires three ingredients to be made: `TOBACCO`, `PAPER` and `MATCHES`. There are three smokers around a table, each of which has an infinite supply of only **one** ingredient.

- **Smoker.** Each smoker is in a loop waiting for both of his missing ingredients to appear on the *table*. Whenever that happens, he grabs the ingredients (the table becomes empty), rolls a cigarette and smokes it by printing a message. A smoker must also put one unit of his own resource on the table whenever asked to do so.
- **Master Agent.** Whenever the table is empty, the *master agent* sends a message demanding a unit of resource to be put on the table to two distinct *smokers* using a channel. The *master agent* chooses the *smokers* that have to put their own resource on the table using a *uniform random distribution*.

Simulate the system and verify that it behaves correctly: infinite execution trace in which each **smoker** smokes infinitely often.

Exercise 8: railway station

Exercise: In a railway station **trains** are countinuously arriving and leaving. Goods are contained in some cargos and, depending on the weight, they are moved from/to either **trucks** or **vans**.
Write a Promela program that models this scenario considering **each cargo as a message** that should be sent/received through the right channel. Each **channel** (train, truck and van) can contain **16 cargos** as a maximum. The **maximum weight** of each cargo in a van is **128**.
You will need two processes:

- `''split''`, that splits goods from the train channel, dividing them over the other two channels, truck and van, depending on the weight values attached
- `''merge''`, that merges the two streams back into one, most likely in a different order, and writes it back into the train channel.

Here are the initial cargo weights on the train: $345, 12, 6777, 32, 0;$

Example 9: word counter

Exercise: In each sentence (string hereafter) the number of the characters composing the string is greater or equal than the number of the words contained in the sentence. A word is characterized by delimiters:

- space ' '
- tabulation '\t'
- endline '\n'

Write a spin function **count()** that perfoms property-based slicing of a string channel, counts the number of characters **nc** and the number of words **nw** and checks if the property $nc >= nw$ is always true.

Use the init function to pass to count() a string (remember that you can model a string as a channel of integers corresponding to ASCII characters).