Spin Tutorial

CS4211 - Formal Methods for Software Engineering

Prepared by: Zhiyu Fan

Install SPIN

There are multiple versions of SPIN available.

You can find the official document here: http://spinroot.com/spin/Man

In this module, you may use three of them

Command line based:

1. Spin

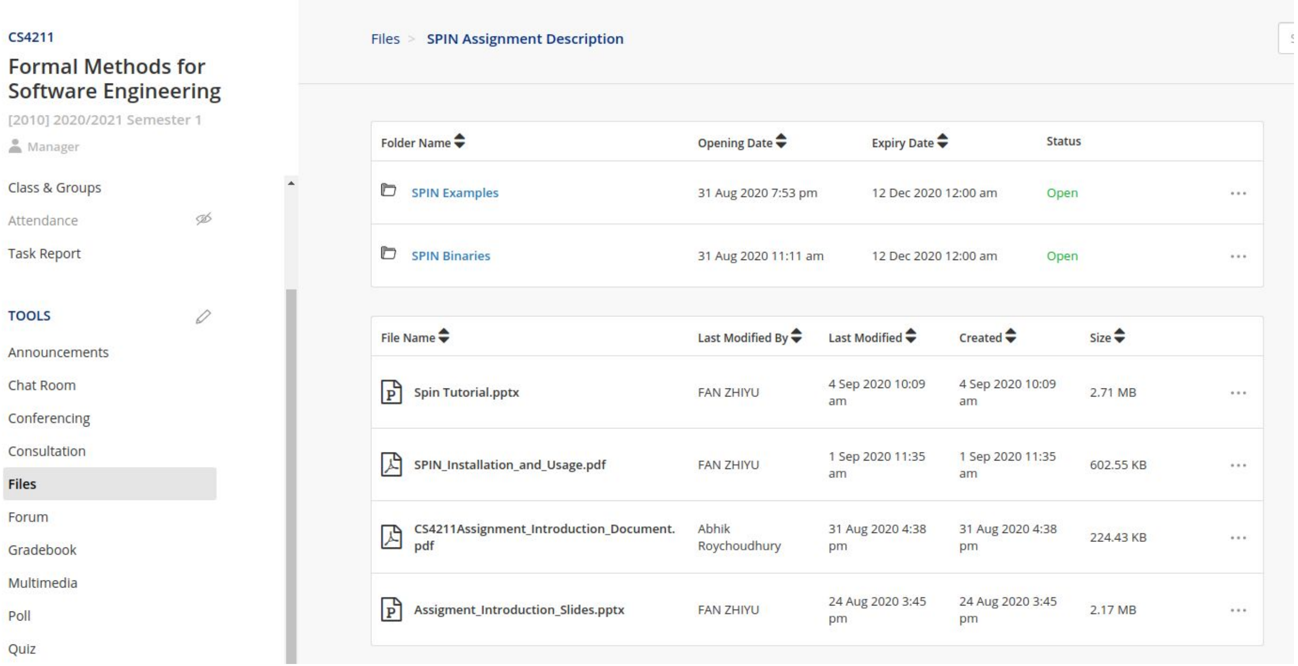
GUI based:

1. iSpin: Written in Tcl/Tk
2. jSpin: Written in Java

Tutorial about how to install those tools have been uploaded to Luminus.

Luminus - Structure

Go to Luminus -> File -> SPIN Assignment Description



Example 1: Hello World

```
active proctype Hello() {
    printf("Hello process, my pid is: %d\n", _pid);
}

init {
    int lastpid;
    printf("init process, my pid is: %d\n", _pid);
    lastpid = run Hello();
    printf("last pid was: %d\n", lastpid);
}
```


NUS
National University of Singapore

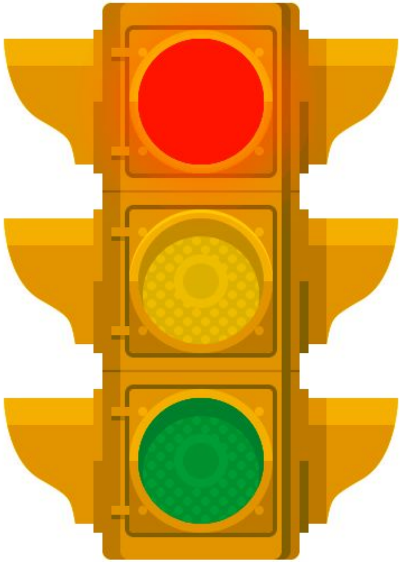
Example 2: Blocked Statement

```
int x = 0;

proctype p1() {
  printf("P1 will be blocked\n");
  x;
  printf("P1 is execuable now\n");
}
```

```
proctype p2() {
  printf("P1 will be released
now\n");
  x = 1;
}

init {
  run p1();
  run p2();
}
```

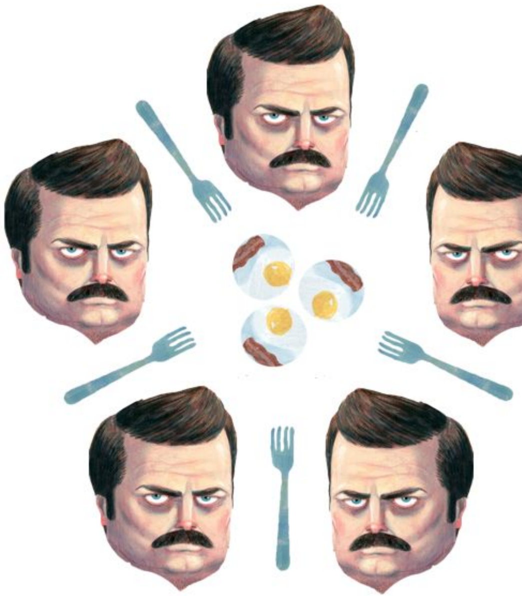
Example 3: Traffic Light Code

```
mtype = { RED, YELLOW, GREEN } ;

active proctype TrafficLight(){

    byte state = GREEN;
    do
    ::    (state == GREEN) -> state = YELLOW;
    ::    (state == YELLOW) -> state = RED;
    ::    (state == RED) -> state = GREEN;
    od;

}
```



Dining Philosophers Problem

The dining philosophers problem is a classic concurrency problem dealing with synchronization.



Source:
http://adit.io/posts/2013-05-11-The-Dining-Philosophers-Problem-With-Ron-Swanson.html

Dining Philosophers Problem

Each philosopher must alternately think and eat. However, a philosopher can only eat spaghetti when they have both left and right forks.

Each fork can be held by only one philosopher and so a philosopher can use the fork only if it is not being used by another philosopher. After an individual philosopher finishes eating, they need to put down both forks so that the forks become available to others.

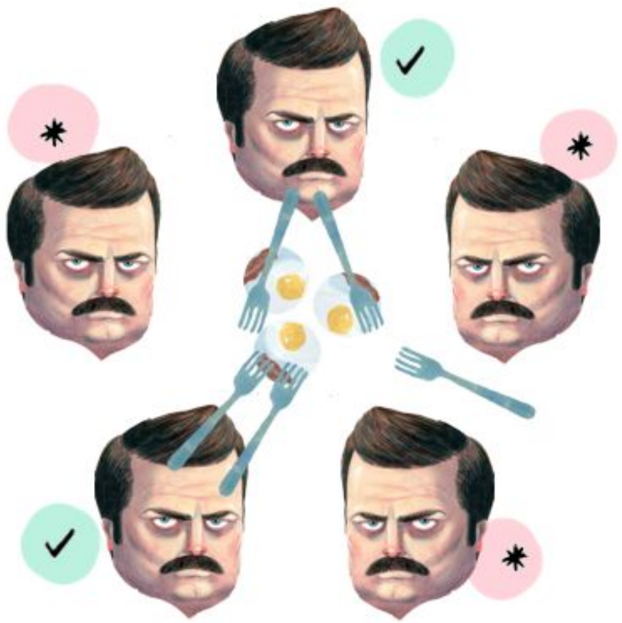
A philosopher can only take the fork on their right or the one on their left as they become available and they cannot start eating before getting both forks.

If Ron gets two forks, he can eat!

If he only has one fork he can't eat :(

Dining Philosophers Problem

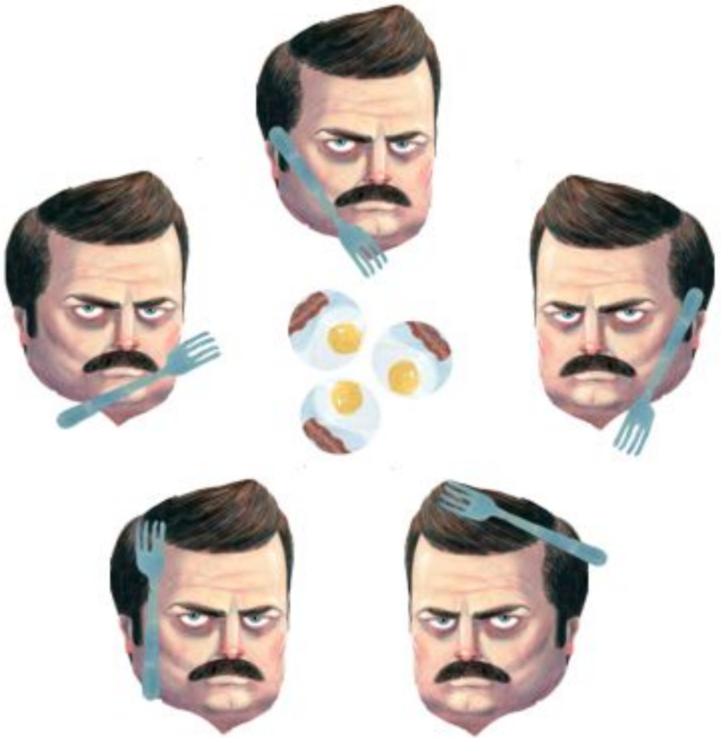
The dining philosophers problem is:
how do you make sure every Philosopher gets to eat?



Dining Philosophers Problem

Every philosopher grabbed a fork: and then they waited for someone to give up their fork so they could eat. But of course, "a Philosopher never gives up his fork!" *sigh* So they waited forever and eventually died in their cabin.

When all philosophers are stuck, that is called ***deadlock***.



Example 4-1: Multiple Process Dining Philosophers

```
int forks = 2;

active [2] proctype Philosophers(){

    forks >0 -> forks = forks -1;
    forks >0 -> forks = forks -1;
    printf("I get to Eat! I am: %d\n", _pid);

}
```

Example 4-2: Mutual Exclusion Dining Philosophers

```
bit forks[2];

active [2] proctype Philosophers(){
    int count, id, bites;
    do
    :: atomic{forks[0] == 0 -> forks[0] = 1; count++; id = 0}
    :: atomic{forks[1] == 0 -> forks[1] = 1; count++; id = 1}
    :: atomic{count == 1 -> forks[id] = 0; count = 0;}
    :: count == 2 -> printf("I get to Eat! I am: %d\n", _pid);
                     count = 0;forks[1]=0; forks[0]=0; bites++;
    od;
}
```


National University of Singapore

Example 5: Alternate Bit Protocol

```
mtype {MSG, ACK};

chan toS = [2] of {mtype, bit};
chan toR = [2] of {mtype, bit};

proctype Sender(chan in, out)
{
  bit sendbit, recvbit;
  do
  :: out ! MSG, sendbit ->
        in ? ACK, recvbit;
        if
        :: recvbit == sendbit ->
           sendbit = 1-sendbit
        :: else
        fi
  od
}
```

channel length of 2

```
proctype Receiver(chan in, out)
{
  bit recvbit;
  do
  :: in ? MSG(recvbit) ->
     out ! ACK(recvbit);
  od
}

init
{
  run Sender(toS, toR);
  run Receiver(toR, toS);
}
```

Alternative notation:

```
ch ! MSG(par1, ...)
ch ? MSG(par1, ...)
```

Example 6: Process Priority

```
byte cnt;

active proctype medium() priority 5{
      set_priority(0, 8);
      printf("medium %d - pid %d pr %d pr1 %d\n",
                _priority,
                _pid,
                get_priority(_pid),
                get_priority(0));
      cnt++
      }
```

```
active proctype high() priority 10{
      _priority = 10;
      printf("high %d\n", _priority);
      cnt++
}

active proctype low() priority 1{
      assert(_priority == 1 && cnt == 2);
      printf("low %d\n", _priority);
}
```


NUS
National University of Singapore

Example 7: Verify LTL

Example from Lecture 4 Slides

EXERCISE

- Two students are taking the CS4211 exam. We must ensure that they cannot leave the exam hall at the same time. To prevent this, each student reads a shared token n before leaving the hall. The shared token is an arbitrary natural number. The global state of the system is given by **<s1, s2, n>** where s1 and s2 are the local states of students 1 and 2 respectively; s1 ∈ {in, out}, s2 ∈{in, out}. The pseudo-code executed by the two students is given below. The two student processes are executed asynchronously. Every time one process is scheduled, it atomically executes one iteration of its loop. Initially s1 =in and s2 = in.

do forever{	do forever{
if (s1 = in & n is odd) {s := out}	if (s2 = in & n is even) {s2:= out}
else if (s1=out) {s1:=in;n:=3*n+1}	else if (s2=out & n is even) {s2:=in;n:=n/2}
else {do nothing}	else {do nothing}
}	}

- Draw the Kripke Structure for this example by maintaining approx info about n. Now state the mutual exclusion property in CTL and check it manually **(by inspection).**

Example 7: Verify LTL

```
chan data = [1] of {int};
int x = 100;
active proctype A() {
  do
  :: data!x -> data?x;
     if
     :: x%2 -> x = 3*x+1;
     :: else
     fi
  od
}
```

```
active proctype B() {
  do
  :: data?x ->
     if
     :: !(x%2) -> x = x/2;
     :: else
     fi
     data!x;
  od
}

#define q (x == 1)
ltl p1 {[](<>q)}
```