Spin Exercises

Behavior models and verification
Recall: Spin

- Explicit state model checker
  - Generates all states of the model to verify
- Input language – Promela
  - Set of processes with interleaving statements
  - Communicating via
    - Global variables
    - Channels
- Finite state models only!
Recall: Example of Promela

```c
bool turn, flag[2];
byte ncrit;

active [2] proctype user()
{
    assert(_pid == 0 || _pid == 1);
    again:
        flag[_pid] = 1;
        turn = _pid;
        (flag[1 - _pid] == 0 || turn == 1 - _pid);
        ncrit++;
        assert(ncrit == 1);
        /* critical section */
        ncrit--;
        flag[_pid] = 0;
        goto again
}
GUI for Spin

- Several implementations
- The best one is **iSpin**
  - Tcl script, TclTk interpreter required
  - For windows I recommend ActiveTcl
  - Be sure to set paths to both spin.exe and gcc.exe (I use cygwin)
Exercise: Producer-Consumer Model

- Describe producer/consumer problem in Promela using channels and check the model for invalid end states (deadlocks) and channels’ buffer overruns
  - i.e., suppose channels are not blocked (messages get lost instead) and you must control the number of messages within the channel by hand
#define SIZE 5
channel c = [6] of {byte};
channel d = [1] of {boolean}
byte fullness = 0;

active proctype producer() {
    byte data;
    do
        :: fullness < SIZE -> fullness = fullness + 1; c!data; data++;
        :: d ? true;
    od
}

active proctype consumer() {
    byte data;
    do
        :: c?data; fullness = fullness - 1; d!true
    od
}

active proctype monitor() {
    assert (fullness <= SIZE);
}
Evaluating Search Complexity – Simulation

How many reachable states do you predict will the following naive Promela model generate?

```promela
init {
  byte i = 0;
  do
    :: i = i + 1;
  od
}
```

$ spin -p -l ex1a.pml$
Now we verify the model:

$ \text{spin} -a \text{ex1a.pml}$
$\text{gcc -o pan pan.c}$
$./pan$
Exercise

Predict how many reachable states there are for the following system. Write them down as a complete reachability tree.

```c
#define N 2
init {
    chan dummy = [N] of { byte };
    do
        :: dummy!85
        :: dummy!170
    od
}
```
Exercise – Evaluation

$ spin -m -a ex1b.pml
  # use -m to ignore buffer overflow
$ gcc -o pan pan.c
$ ./pan
What happens if you set N to 3? Express the number of states as a function of N. Use the formula to calculate how many states there will be if you set N to 14? Check your prediction:

```
$ spin -m -a ex1b.pml
$ gcc -o pan pan.c
$ ./pan
```
Comments on Memory usage I.

- The efficiency of the conventional reachability analysis is determined by the state space storage functions. To study this, repeat the last verification run with a smaller and a bigger hash table for storing reachable states:

  $ \text{pan -w10} \ # \text{hash table with } 2^{10} \text{ slots} \ldots$
  $\text{pan -w20} \ # \text{hash table with } 2^{20} \text{ slots} \ldots$
Comments on Memory usage II.

- Bit-state hashing method
  - Probabilistic approach
  - Uses all available (specified) memory
    - Might miss some states

$ spin -m -a ex.1b.pml # as before
$ gcc -DBITSTATE -o pan pan.c # different
$ ./pan
Questions? / Dotazy?