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 and channels

Finite state models only!
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;
}
```
Several implementations

The best one (and sort-of official) 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 used cygwin)
EVALUATING SEARCH COMPLEXITY – SIMULATION

How many reachable states does the following naïve Promela model generate?

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

$ spin -p -l ex1a.pml$
EVALUATING SEARCH COMPLEXITY – VERIFICATION

Now we verify the model:

$ spin -a ex1a.pml
$ gcc -o pan pan.c
$ ./pan
Estimate how many reachable states there are for the following model. Draw the complete reachability tree.

```c
#define N 2
init {
    chan dummy = [N] of { byte };
    do
        :: dummy!85
        :: dummy!170
    od
}
```
$ spin -m -a ex1b.pml  # use -m to ignore buffer overflow
$ gcc -o pan pan.c
$ ./pan
EXERCISE – CONTD.

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
```
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:

```bash
$ pan -w10 # hash table with 210 slots ...
$ pan -w20 # hash table with 220 slots ...
```
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
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