A very short introduction to Spin

Franco Raimondi

Department of Computer Science
School of Science and Technology
Middlesex University
http://www.rmnd.net

A very short introduction to Spin

Today's menu

You should be able to translate the question "Does system S satisfy the property P?" into the formal statement: $M_S \models \varphi_P$, for the temporal logic LTL.

Aims of this lecture:

- Define M_S using a dedicated (and simple) programming language.
- Use a tool (*Model Checker*) to verify whether or not $M_S \models \varphi_P$.

Review of LTL Syntax

$$\varphi ::= \top \mid p \mid \varphi \wedge \varphi \mid \neg\varphi \mid \bigcirc \varphi \mid \varphi U \varphi$$

Derived operators:

- $\Diamond\varphi \equiv \top U \varphi$
- $\Box\varphi \equiv \neg\Diamond\neg\varphi$

I will use also: X, F, G for $\bigcirc, \Diamond, \Box$.

Review of LTL Semantics

Let σ be a word over 2^{AP}. This means that $\sigma = A_0 A_1 \ldots$, where each A_i is a set of atomic propositions.
Satisfaction is a relation $\models (2^{AP})^\omega \times \mathrm{LTL}$:

$$
\begin{array}{lll}
\sigma \models \top & & \\
\sigma \models p & \text{iff} & p \in A_0 \\
\sigma \models \varphi \wedge \psi & \text{iff} & \sigma \models \varphi \text{ and } \sigma \models \psi \\
\sigma \models X\varphi & \text{iff} & \sigma[1:] \models \varphi \\
\sigma \models \varphi U \psi & \text{iff} & \exists i \geq 0 \text{ s.t. } \sigma[i:] \models \psi \\
& & \text{and } \forall 0 \leq j < i, \sigma[j:] \models \varphi
\end{array}
$$

LTL Semantics

- $s \models \varphi$ if Traces(s) $\models \varphi$ (i.e., each trace satisfies φ).
- $TS \models \varphi$ iff $s_0 \models \varphi$ for all $s_0 \in I$.

Spin: overview

- Spin (= Simple Promela INterpreter) targets the efficient verification of multi-threaded software (NuSMV originally for hardware circuits).
- Provides direct support for the use of embedded C code as part of model specifications. Promela (= Process Meta Language) code can also be generated from C code (see modex).
- Can use multi-core machines and works *on-the-fly*.
- Source and binaries available from `http://spinroot.com`.

Promela basics

At a very high level, the structure of a Promela file is:

```
mtype = ... /* a list of message types */
chan someName == ... /*example channel */
bool something; /* a global var. */

proctype Process1() { /*a process */
   ...
}

proctype Process2() { /* another one */
   ...
}

init { /* optional init section */
   ...
}
```

Example: Peterson

```
#define p (ncrit==1)

bool turn, flag[2]; // the shared variables, booleans
byte ncrit;         // nr of procs in critical section

active [2] proctype user() // two processes
{
 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
}

ltl p1 {<>[] !p}
ltl p2 {[]<> !p}
```

Command-line Spin

```
$ ./spin645_mac -a peterson.pml
$ gcc -o pan pan.c
$ ./pan -a -N p2
```

Check the output:

```
[...]
State-vector 36 byte, depth reached 49, errors: 0
```

Now try with p1:

```
[...]
State-vector 36 byte, depth reached 49, errors: 1
[...]
pan: wrote peterson.pml.trail
```