SPIN: Part 2

15-414 Bug Catching: Automated Program Verification and Testing

Sagar Chaki
November 2, 2011

Control flow

We have already seen some

- Concatenation of statements, parallel execution, atomic sequences

There are a few more

- Case selection, repetition, unconditional jumps

Case selection

```
if
:: (a < b) → option1
:: (a > b) → option2
:: else → option3                    /* optional */
fi
```

Cases need not be exhaustive or mutually exclusive

- Non-deterministic selection

Repetition

```
byte count = 1;
proctype counter() {
        do
        :: count = count + 1
        :: count = count – 1
        :: (count == 0) → break
        od
}
```

Repetition

```
proctype counter()
{
        do
        :: (count != 0) →
                if
                :: count = count + 1
                :: count = count – 1
                fi
        :: (count == 0) → break
        od
}
```

Unconditional jumps

```
proctype Euclid (int x, y)
{
        do
        :: (x > y) → x = x – y
        :: (x < y) → y = y – x
        :: (x == y) → goto done
        od ;
        done:  skip
}
```

Procedures and Recursion

Procedures can be modeled as processes

- Even recursive ones
- Return values can be passed back to the calling process via a global variable or a message

Time for example 3

Timeouts

```
Proctype watchdog() {
    do
    :: timeout → guard!reset
    od
}
```

Get enabled when the entire system is deadlocked

No absolute timing considerations

Assertions

assert(any_boolean_condition)

- pure expression

If condition holds ⇒ no effect

If condition does not hold ⇒ error report during verification with Spin

Time for example 4

LTL model checking

Two ways to do it

Convert Kripke to Buchi

- Convert claim (LTL) to Buchi
- Check language inclusion

OR

- Convert ~Claim (LTL) to Buchi
- Check empty intersection

What Spin does

Checks non-empty intersection

- Requires very little space in best case

Works directly with Promela

- No conversion to Kripke or Buchi

Must provide Spin with negation of property you want to prove

LTL syntax in SPIN

ϕ :=	p		proposition
	|		true
	|		false
	|		(ϕ)
	|		ϕ binop ϕ
	|		unop ϕ

unop :=	[]	always (G)
	| <>	eventually (F)
	| X	next time
	| !	logical negation
binop :=	U	strong until
	| &&	logical AND
	| ||	logical OR
	| ->	implication
	| <->	equivalence

Time for example 5

Peterson's Algorithm in SPIN

```
bool turn, flag[2];

active [2] proctype user()
{
  assert(_pid == 0 || _pid == 1);
again:

= 0 || turn == 1 - _pid);

            /* critical section */

  flag[_pid] = 0;
  goto again;
}
```

Active process:

automatically creates instances of processes

_pid:

Identifier of the process

assert:

Checks that there are only at most two instances with identifiers 0 and 1

Peterson's Algorithm in SPIN

```
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;
}
```

ncrit:

Counts the number of

Process in the critical section

assert:

Checks that there are always

at most one process in the

critical section

Peterson's Algorithm in SPIN

```
bool turn, flag[2];
bool critical[2];

active [2] proctype user()
{
  assert(_pid == 0 || _pid == 1);
again:
  flag[_pid] = 1;
  turn = _pid;
  (flag[1 - _pid] == 0 || turn == 1 - _pid);

  critical[_pid] = 1;
  /* critical section */
  critical[_pid] = 0;

  flag[_pid] = 0;
  goto again;
}
```

LTL Properties:

mutex

1. [] (!critical[0] || !critical[1])

no starvation

2. []<> (critical[0]) && []<> (critical[1])

alternation

3. [] (critical[0] -> (critical[0] U (!critical[0] && ((!critical[0] && !critical[1]) U critical[1]))))

alternation

4. [] (critical[1] -> (critical[1] U (!critical[1] && ((!critical[1] && !critical[0]) U critical[0]))))

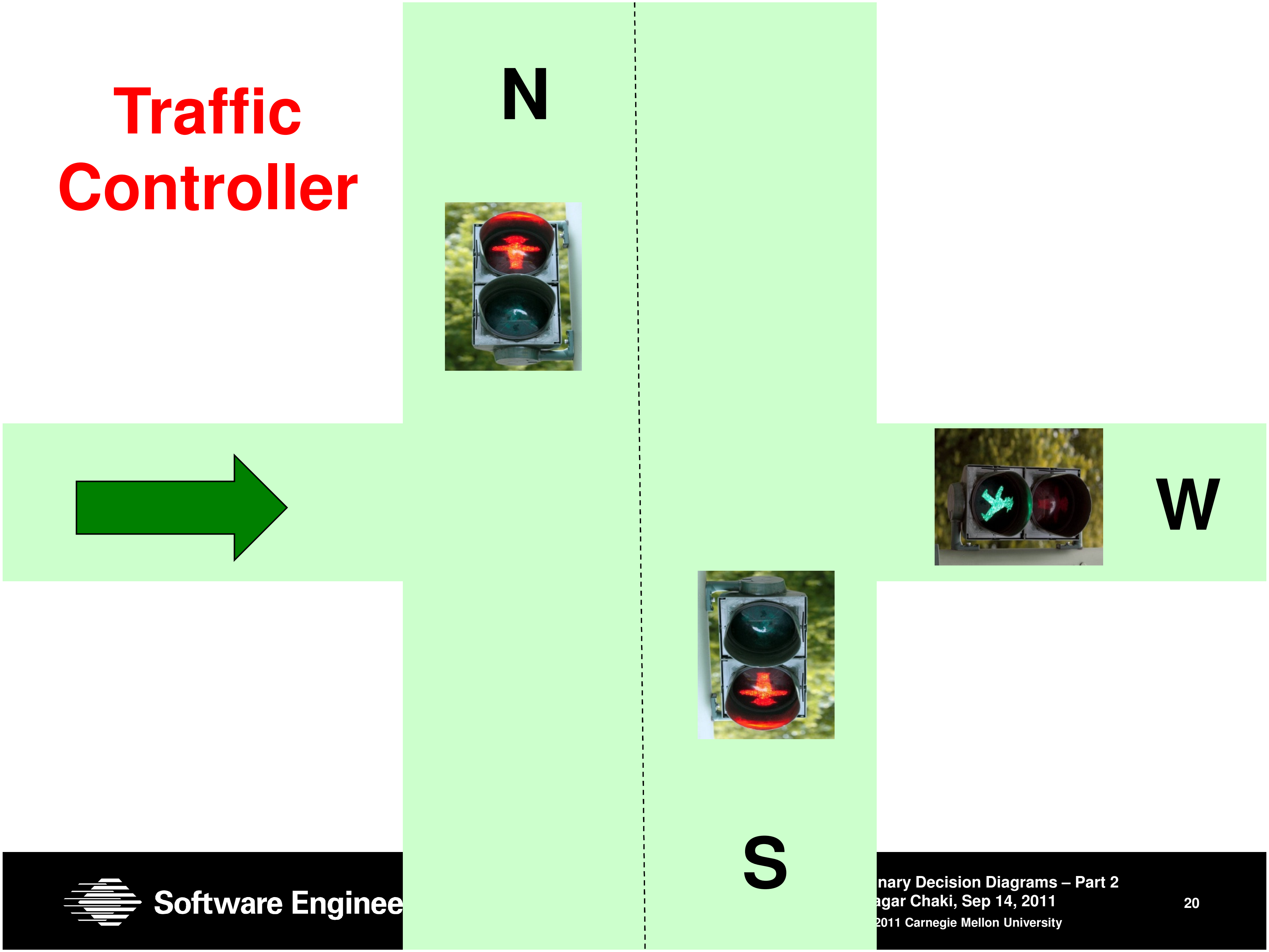
Mutual Exclusion in SPIN

```
bool turn, flag[2];
bool critical[2];

active [2] proctype user()
{
  assert(_pid == 0 || _pid == 1);
again:
  flag[_pid] = 1;
  turn = _pid;
  (flag[1 - _pid] == 0 || turn == 1 - _pid);

  critical[_pid] = 1;
  /* critical section */
  critical[_pid] = 0;

  flag[_pid] = 0;
  goto again;
}
```

LTL Properties (negated):

1. <> (critial[0] && critical[1]) — holds
2. <>[] (!critical[0]) || <>[] (!critical[1]) — holds
3. <> (critical[0] && !(critical[0] U (!critical[0] && ((!critical[0] && !critical[1]) U critical[1])))) — does not hold
4. <> (critical[1] && !(critical[1] U (!critical[1] && ((!critical[1] && !critical[0]) U critical[0])))) — does not hold

Traffic Controller



Modeling in SPIN

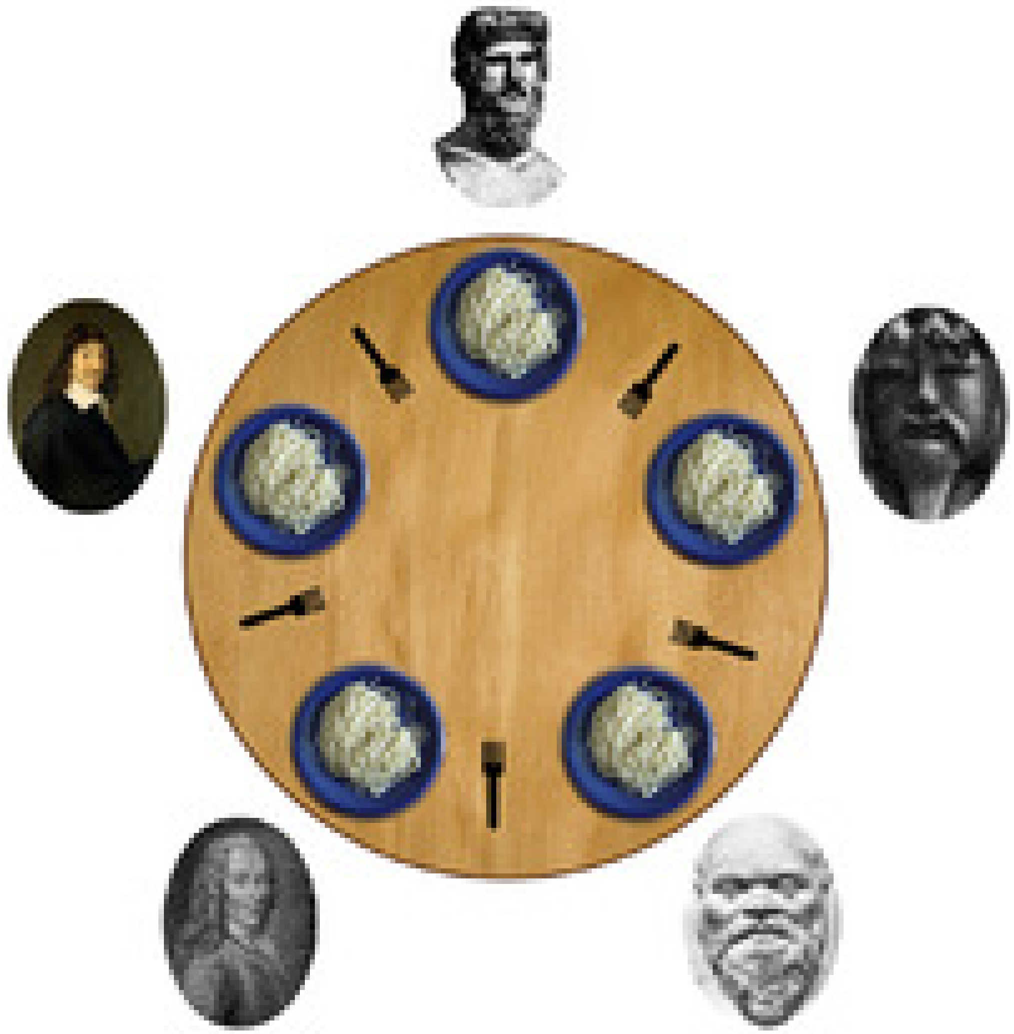
System

- No turning allowed
- Traffic either flows East-West or North-South
- Traffic Sensors in each direction to detect waiting vehicles
- Traffic.pml

Properties:

- Safety : no collision (traffic1.ltl)
- Progress – each waiting car eventually gets to go (traffic2.ltl)
- Optimality – light only turns green if there is traffic (traffic3.ltl)

Dining Philosophers



Modeling in SPIN

Each fork is a rendezvous channel

A philosopher picks up a fork by sending a message to the fork.

A philosopher releases a fork by receiving a message from the fork.

Properties

- No deadlock
- Safety – two adjacent philosophers never eat at the same time – dp0.ltl
- No livelock – dp1.ltl
- No starvation – dp2.ltl

Versions

- dp.pml – deadlock, livelock and starvation
- dp_no_deadlock1.pml – livelock and starvation
- dp_no_deadlock2.pml – starvation

References

http://cm.bell-labs.com/cm/cs/what/spin/

http://cm.bell-labs.com/cm/cs/what/spin/Man/Manual.html

http://cm.bell-labs.com/cm/cs/what/spin/Man/Quick.html

Questions?

Sagar Chaki
Senior Member of Technical Staff
RTSS Program
Telephone: +1 412-268-1436
Email: chaki@sei.cmu.edu

U.S. Mail
Software Engineering Institute
Customer Relations
4500 Fifth Avenue
Pittsburgh, PA 15213-2612
USA

Web
www.sei.cmu.edu/staff/chaki

Customer Relations
Email: info@sei.cmu.edu
Telephone: +1 412-268-5800
SEI Phone: +1 412-268-5800
SEI Fax: +1 412-268-6257