Cockrell School of Engineering

Verifying Distributed Algorithms
in Promela

Eric Crosson
Nhan Do
Stormy Mauldin
Daniel Officewala

EE 360P
April 28, 2016

Outline

Introducing Promela

Dining Philosophers

Token Ring

Lamport’s

Szymanski’s

Questions

What are Spin and Promela?

Spin is a multi-threaded software verification tool that supports the high-level language Promela to specify models of systems.

- Promela allows the user to create parallel programs that define processes, message channels, and variables.
- From the promela program, Spin can then verify various system correctness properties as well as user-defined properties

Dining Philosophers

Most Philosophers

- Want to eat:
  - Get left fork (or wait)
  - Get right fork (or wait)
- After eating:
  - Release both forks
  - Contemplate life until hungry

One "Special" Philosopher

- Want to eat:
  - Get left fork (or wait)
  - Get right fork (if fail, release *both* forks)
- After eating:
  - Release both forks
  - Contemplate life until hungry

Dining Philosopher Analysis

- **Mutually Exclusive**
  - A philospher must acquire both shared forks before eating
- **Deadlock**
  - The one special philosopher will always surrender both forks, allowing someone else to start eating.
- **Starvation**
  - The special philosopher always surrenders forks in case of conflict. May never get a change to eat.

Figure: Hungry, hungry philosophers

Token Ring Algorithm

- Simple and easily scalable
  - Pass token around ring of processes
  - Only processes with token can enter CS
  - No Starvation

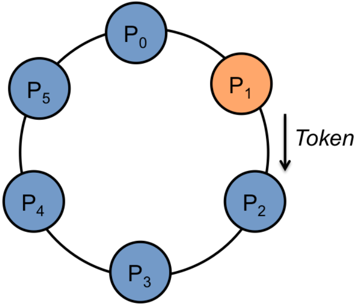


Figure: Token Ring Algorithm

Token Ring Implementation

```
#ifndef N
#define N 10
#endif

#ifndef size
#define size 10
#endif

byte state[N];
bit _Permission[N];
bit _Executing[N];

byte in_cs;
byte token;

init {
  atomic {
    byte i = 0;
        token = 0;
```

```
    do
    :: i < N –> run P(i); i++;
    :: else  –> break;
    od;
  }
}


proctype P(byte id) {
NonCritical:
  _Permission[id] = true;
  printf("Token value: %d", token);

Wait:
  _Executing[id] = true;
  if
  :: id != token –> _Permission[id] = false; goto Wait;
  :: id == token
  fi;
```

```
  if
  :: _Permission[id] == false –> goto
  :: atomic { _Permission[id] == true
  fi;
Critical:
        atomic { in_cs––; }
       printf("Process %d has entere
       _Permission[id] = false;
       _Executing[id] = false;
       if
    :: token < N –> token = ((token
       :: atomic{token > (N–1) –> to
}
    fi;
       goto NonCritical;
}
```

Lamport's

An equation block

$$\vec{F} = m\vec{a}$$

- Second instance of citations use short citation hyperlinked to original.

Szymanski's Algorithm

- Extension of Lamport's
  - satisfies linear wait
  - three booleans per process
- Extension of Lamport's

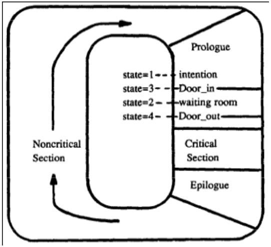
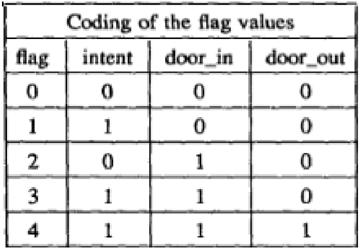


Figure: Szymanski's Algorithm



Coding of the flag values			
flag	intent	door_in	door_out
0	0	0	0
1	1	0	0
2	0	1	0
3	1	1	0
4	1	1	1

Figure: State-tracking booleans

Szymanski’s Implementation

```
start:
    /* 1. SEKCJA LOKALNA */

    local_section();

    /* 2. PROLOG */
    intent[i] = true;

started_protocol:
    skip;

    /* 3. Others are trying to
     * enter waiting room? */
    (count(1,1,0) +
      count(1,1,1) == 0);

    /* 4. Enter waiting room */
    door_in[i] = true;
```

```
anteroom_check:
    if
      :: (count(1,0,0) +
            count(1,0,1) > 0) ->
        {
            /* State 2  */
            intent[i] = false;

        in_anteroom:
            ((count(0,0,1) +
              count(0,1,1) +
              count(1,0,1) +
              count(1,1,1) > 0)
            );

            /* State 3 */
            intent[i] = true;
        }
      :: else
    fi;
```

```
    /* . Proceed into CS when
     * it is your turn */
    door_out[i] = true;

    wait_forall(k, i + 1, N,
      (!door_in[k] || door_out[k]));

    wait_forall(k, 0, i,
      (!door_in[k]));

critical_section:
    /* . SEKCJA KRYTYCZNA */

    critical_section();

    /* . EPILOG */
    door_out[i] = false;
    door_in[i] = false;
    intent[i] = false;
```

Szymanski’s Analysis

Questions

- English mothafucka, do you speak it?