Cockrell School of Engineering

Verifying Distributed Algorithms
in Promela

Eric Crosson
Nhan Do
Stormy Mauldin
Daniel Officewala
EE 360P
June 21, 2016

Outline

- Promela Overview
- Dining Philosophers
- Token Ring
- Chandy and Lamport
- Szymanski's
- Questions

Promela Overview

- Promela is **Pr**ocess **Me**ta **La**nguage
- Spin compiles Promela into C
- C executables assert system invariants at each simulated state

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.

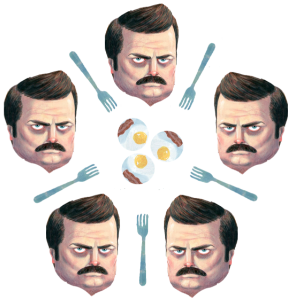

Figure: Hungry, hungry philosophers

Token Ring Algorithm

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

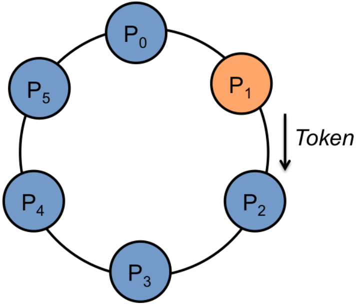

Figure: Token Ring Algorithm

Token Ring Implementation

```
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;

Wait:
  _Executing[id] = true;
  if
  :: id != token ->
        _Permission[id] = false;
        goto Wait;
  :: id == token
  fi;
  if
  :: _Permission[id] == false ->
        goto NonCritical;
  :: atomic { _Permission[id] ==
```

```
        true -> in_cs++; }
    fi;
 Critical:
        atomic { in_cs--; }
        _Permission[id] = false;
        _Executing[id] = false;
        if
    :: token < N ->
        token = ((token + 1) % N);
        :: atomic{token > (N-1) ->
        token = 0;  }
    fi;
        goto NonCritical;
```

Chandy and Lamport

- Gurarantees consistent global snapshots
  - Happened-before model
  - Uses markers
  - Promela model verification based on markers

Chandy Implementation

```
mtype { message, marker };
chan ch = [N] of
      { mtype, byte };

active proctype Sender() {
  do
  :: lastSent < NUM_MESSAGES ->
       lastSent++;
       ch ! message(lastSent)
  :: ch ! marker(0) ->
       break
  od }
```

```
active proctype Receiver() {
  byte received;
  do
  :: ch ? message(received) ->
        lastReceived = received
  :: ch ? marker(_) ->
        messageAtMarker =
          lastReceived;
        if
        :: !recorded ->
            messageAtRecord =
              lastReceived
```

```
        :: else
        fi;
        break
  :: !recorded ->
        messageAtRecord =
          lastReceived;
        recorded = true
  od
}
```

Szymanski's Algorithm

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

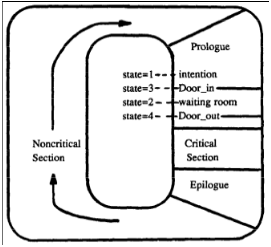

Figure: Szymanski's Algorithm

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;
```

Questions

Thank you for your time.