CS477 Formal Software Development Methods

Elsa L Gunter
2112 SC, UIUC
egunter@illinois.edu
http://courses.engr.illinois.edu/cs477

Slides mostly a reproduction of Theo C. Ruys – SPIN Beginners’ Tutorial

April 6, 2018
LTL Model Checking

- **Model Checking Problem**: Given model \( M \) and logical property \( \varphi \) of \( M \), does \( M \models \varphi \)?

- Given transition system with states \( Q \), transition relation \( \delta \) and initial state state \( I \), say \( (Q, \delta, I) \models \varphi \) for LTL formula \( \varphi \) if every run of \( (Q, \delta, I) \), \( \sigma \) satisfies \( \sigma \models \varphi \).

**Theorem**

*The Model Checking Problem for finite transition systems and LTL formulae is decidable.*

- Treat states \( q \in Q \) as letters in an alphabet.
- Language of \( (Q, \delta, I) \), \( L(Q, \delta, I) \) (or \( L(Q) \) for short) is set of runs in \( Q \)
- Language of \( \varphi \), \( L \varphi = \{ \sigma | \sigma \models \varphi \} \)
- Question: \( L(Q) \subseteq L(\varphi) \)?
- Same as: \( L(Q) \cap L(\neg \varphi) = \emptyset \)?
How to Decide the Model Checking Problem?

- How to answer $\mathcal{L}(Q) \cap \mathcal{L}(\neg \varphi) = \emptyset$?
- Common approach:
  - Build automaton $A$ such the $\mathcal{L}(A) = \mathcal{L}(Q) \cap \mathcal{L}(\neg \varphi)$
  - Are accepting states of $A$ reachable? (Infinitely often?)
- How to build $A$?
  - One possible answer: Build a series of automata by recursion on structure of $\neg \varphi$.
  - Another possible answer: Build an automaton $B$ such $\mathcal{L}(B) = \mathcal{L}(\neg \varphi)$; take $A = B \times Q$
- Will do at least one approach if time after Spin
Introduction to SPIN and Promela

- SPIN Background
- Promela processes
- Promela statements
- Promela communication primitives
- Architecture of (X)Spin
- Some SPIN demo’s
  - hello world
  - mutual exclusion
  - alternating bit protocol

Slides based heavily on: Theo C. Ruys - SPIN Beginners’ Tutorial
SPIN Documentation

**SPIN Overview**

- **Input:**
  - (Abstract) model of system
  - Behavior specification

- **Output:**
  - Says whether model satisfies specification
  - If models fail specification, give a system run that violates requirement (counterexample)

- Focused on correctness of process communications and interactions

- Internal details generally abstracted away
SPIN Introduction

SPIN = Simple Promela Interpreter
- Tool for analyzing logical consistency of concurrent systems
  - specifically data communication protocols
- state-of-the-art model checkers, thousands of users
- Concurrent systems described in modelling language Promela

Promela = Protocol/Process Meta Language
- Resembles C programming language
- Supports dynamic creation of concurrent processes
- limited to describing finite-state systems
- Communication via message channels
  - Synchronous (rendezvous)
  - Asynchronous (buffered)
Promela Models

Promela model consist of:

- type declarations
- channel declarations
- variable declarations
- process declarations
- [init process]

A Promela model corresponds with a (usually very large, but) finite 
transition system, so

- no unbounded data
- no unbounded channels
- no unbounded processes
- no unbounded process creation
promela Skeleton Example

mtype = \{MSG, ACK\};
chan toS = ...
chan toP = ...
bool flag;

proctype Sender() {
  ...
} /* process body */

proctype Receiver() {
  ...
} /* process body */

init {
  ...
} /* creates processes */
A process type (proctype) consists of

- a name
- a list of formal parameters
- local variable declarations
- body consisting of a sequence of statements
proctype *Sender* (chan in; chan out) {
    bit sndB, rcvB;      /* local variables */
    do                     /* body beginning */
        :: out ! MSG, sndB ->
            in ? ACK, rcvB;
        if
        :: sndB == rcvB -> sndB = 1-sndB
        :: else -> skip
        fi
    od                     /* body end */
}
A process

- is defined by a proctype definition
- executes concurrently with all other processes, independent of speed of behaviour
- communicates with other processes
  - using global (shared) variables
  - using channels

May be several processes of the same type

Each process has own local state:

- process counter (location within the proctype)
- contents of the local variables
Process Creation

- Processes created with `run` statement
  - Returns process id
- Process created at any point in execution (of any process)
- Processes start after execution of `run` statement
- Also created by `active` keyword before `proctype` declaration
Sample Proctype Declaration Skeleton

proctype Foo(byte x) {
    ...}

active[3] proctype Bar(byte y) { /* [3] opt; y init to 0 */
    ...}

init {
    int pid2 = run Foo(2);
    run Bar(17);
    run Foo (27);
}
active proctype Hello() {
    printf("Hello process, my pid is: \%d\n", _pid);
}

init {
    int lastpid;
    printf("init process, my pid is: \%d\n", _pid);
    lastpid = run Hello();
    printf("last pid was: \%d\n", lastpid);
}
bash-3.2$ spin hello.pml
   init process, my pid is: 1
   Hello process, my pid is: 0
   Hello process, my pid is: 2
   last pid was: 2
3 processes created
bash-3.2$ spin hello.pml
   Hello process, my pid is: 0
   init process, my pid is: 1
   last pid was: 2
   Hello process, my pid is: 2
3 processes created
Hello Processes

Hello()

init()

print "init"

run Hello()

print "last"

Hello()

print "Hello"
Hello Processes Interleavings

Hello()
p
print "Hello"

init()

run Hello()
p
print "init"
p
print "last"

Hello()
p
print "Hello"
Interleaving Semantics

- **Promela processes** execute **concurrently**.
- **Non-deterministic** scheduling of the processes.
- Processes are **interleaved**
  - Only one process can execute a statement at each point in time.
  - Exception: **rendez-vous communication**.
- All statements are **atomic**
  - Each statement is executed without interleaving it parts with other processes.
- Each process may have several **different possible actions** enabled at each point of execution.
  - Only one choice is made, **non-deterministically** (randomly).