SWEN-220
Mathematical Models of Software

Introduction to SPIN/Promela
Topics

• Introduction to Promela
• Simple Promela program
• Promela basics
  – Data types
  – Operators and expressions
  – if statements
What we learned so far...

• Models are written to describe the design of the system

• Specify correctness properties

• Models are checked for correctness

• Now: model behavior
Checking for Correctness

• Trivial: simply check all possible states of a program and check correct in each state.

• Non-trivial programs have billions, or even trillions possible states.
  – Consider the program that has one 32-bit variable. There may $2^{32}$ different states.

• We need to create models that represent programs, but can be checked for correctness given available resources.
What is Promela? What is SPIN?

- Promela – *Process meta-language*
- Promela is a simple *language* to model concurrent systems

- SPIN – *Simple Promela IN terpreter*
- SPIN is a *model checker* that verifies these models
- We will be using *jSpin*
History

• History: Gerard Holzmann
  – 1980s and 90s at Bell Labs
  – Initially for protocol verification.

• Currently @ JPL; spin for spacecraft verification

• Examples:
  – Dutch flood control algorithms
  – Telephone switch call processing
  – Cassini, Mars Rover, etc.
  – Medical device transmission protocols
Processes

• Programs in Promela are composed of a set of processes

```c
active proctype P()
{
    printf("Hello world\n")
}
```
Example

active proctype P()
{
    int value = 123;
    int reversed;
    reversed = (value % 10)*100 +
                ((value/10)%10)*10 +
                (value/100);
    printf("value = %d, reversed = %d\n", value, reversed)
}
## Data Types

<table>
<thead>
<tr>
<th>Type</th>
<th>Value</th>
<th>Size (bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>bit, bool</td>
<td>0, 1, false, true</td>
<td>1</td>
</tr>
<tr>
<td>byte</td>
<td>0..255</td>
<td>8</td>
</tr>
<tr>
<td>short</td>
<td>-32768…32767</td>
<td>16</td>
</tr>
<tr>
<td>int</td>
<td>-2^{31} .. 2^{31}-1</td>
<td>32</td>
</tr>
<tr>
<td>unsigned</td>
<td>0..2^n-1</td>
<td>&lt;=32</td>
</tr>
</tbody>
</table>
Data Types

- Important to use as few bits as possible to avoid combinatorial explosion
  - *The general rule: "smaller is better"

- All variables initialize to zero by default

- No character type in Promela

- No string variables in Promela

- No floating-point data types
Symbolic Names

• Can use \texttt{mtype} to give a mnemonic or symbolic name

• Can only have a max of 255 values

• There is only one mtype set defined for the entire program.
Operators and expressions

• Operators include
  +,-,*,/,% arithmetic
  &&, ||  logical and, or
  ( - > : ) conditional expression
  =  assignment

• We will cover more operators as needed
Example – Traffic Light

mtype = {red, yellow, green};
mtype light = green;

active proctype P()
{
do
:: if
  :: light == red -> light=green
  :: light == yellow -> light = red
  :: light == green -> light = yellow
  fi;
printf ("The light is now %e\n", light)
od
}
if (control) Statements

• Statement starts with keyword if, ends with fi

• In between there are alternatives, consisting of a guarded command, (or just “guard”), an arrow and sequence of statements

• else guard is executed iff all other guards evaluate to false

• There is no semantic meaning to the order of alternatives in if statements – i.e. the order the guards are in.
Simple Guard Example

• Which branch will be executed?

```c
active proctype P()
{
    int a = 5, b=5;
    int max;
    int branch;
    if
    :: a >= b -> max = a; branch =1
    :: b >= a -> max = b; branch =2
    :: else -> branch = 3
    fi;

    printf("The max of %d and %d = %d by branch %d\n", a, b, max, branch)
}"
```
Guard Statements

• Whenever there exists a nondeterministic choice, SPIN will randomly choose one

• This demonstrates the concept of random simulation
Question

• What color is the printed light?

```c
mtype = {red, yellow, green};
mtype light = green;

active proctype P()
{
  do
    :: if
      :: light == green -> light=green
      :: light == green -> light = red
      :: light == green -> light = yellow
    fi;
  printf ("The light is now %e\n", light)
  od
}
```
DATA TYPES & STRUCTURES - DECLARATIONS

• Global vs. Local.
• Local declarations "promoted" to front of proctype.
• Example:

```c
active proctype P() {
    byte a = 1 ;
    a = 20 ;
    byte b = a ;
    printf("a = %d b = %d\n", a, b) ; /* what is printed? */
}
```
SELECTION (1)

• Non-deterministic if

  if
  :: (x <= y) -> min = x ;
  :: (x >= y) -> min = y ;
  fi
SELECTION (1)

• Non-deterministic if
  
  if
  :: (x <= y) -> min = x ;
  :: (x >= y) -> min = y ;
  fi /* Blocks if neither are true */

• Optional else
  
  if
  :: (x <= y) -> min = x ;
  :: else -> min = y ; /* Non-blocking version */
  fi
SELECTION (2)

• How about:

```
min = y;
if :: x < y -> min = x;
fi
```
Selection (2)

• How about:
  \[\text{min} = y;\]
  \[\text{if}\]
  \[:: x < y \rightarrow \text{min} = x;\]
  \[\text{fi}\]

• Process will \textbf{block} until at least one guard is true.
  \[\text{min} = y;\]
  \[\text{if}\]
  \[:: x < y \rightarrow \text{min} = x;\]
  \[:: \text{true} \rightarrow \text{skip};\]
  \[\text{fi}\]
How about:

```plaintext
min = y;
if :: x < y -> min = x;
fi
```

Process will *block* until at least one guard is true.

```plaintext
min = y;
if :: x < y -> min = x;
:: else -> skip;
fi
```
ITERATION

• GCD of x and y:
  
  \[
  \text{do} \\
  \quad \text{:: } x == y \to \text{ gcd } = x ; \textbf{break} ; \\
  \quad \text{:: } x < y \to \text{ gcd } = y - x ; \; y = x ; \; x = \text{gcd} ; \\
  \quad \text{:: } y < x \to \text{ gcd } = x - y ; \; x = y ; \; y = \text{gcd} ; \\
  \text{od}
  \]
ITERATION

• GCD of x and y:
  do
  :: x == y -> gcd = x ; break ;
  :: x < y -> gcd = y - x ; y = x ; x = gcd ;
  :: y < x -> gcd = x - y ; x = y ; y = gcd ;
  od

• What about:
  do
  :: x == y -> gcd = x ; break ;
  :: x <= y -> gcd = y - x ; y = x ; x = gcd ;
  :: y <= x -> gcd = x - y ; x = y ; y = gcd ;
  od
MisCELLANY

• -> and ; do
  :: x == y ; gcd = x ; break ;
  :: x < y ; gcd = y - x ; y = x ; x = gcd ;
  :: y < x ; gcd = x - y ; x = y ; y = gcd ;
  od
- **and**
  ```
  do
  :: x == y ;  gcd = x ;  break ;
  :: x < y ;  gcd = y - x ;  y = x ;  x = gcd ;
  :: y < x ;  gcd = x - y ;  x = y ;  y = gcd ;
  od

  Else with **do**
  ```
  ```
  do
  :: x < y - >  gcd = y - x ;  y = x ;  x = gcd ;
  :: y < x - >  gcd = x - y ;  x = y ;  y = gcd ;
  :: else - >  gcd = x ;  break ;
  od
  ```
EXECUTABILITY

• Body of a **proctype** contains **statements** and **expressions**.

• Statements are assignments, ++, -- (that is, they change the state).

• Expressions are just values (no state change).

• A statement is **always** executable.

• An expression is executable if it is **true** (\(!= 0\)).
EXECUTABILITY EXAMPLES (1)

if
  :: (x > y) -> x-- ;
  :: (y <= x) -> y++;
  :: true -> x = y + x ;
fi
EXECUTABILITY EXAMPLES (1)

if
:: (x > y) -> x--; 
:: (y <= x) -> y++; 
:: true -> x = y + x ; /* true is always an option to be selected */
fi

is equivalent to

if
:: (x > y) -> x--; 
:: (y <= x) -> y++; 
:: x = y + x ; /* as a guard, this evaluates to “1” – true */
fi
Wait for another process to set *proceed* to true:

\[
\begin{align*}
x &= y \\
do \\
::& \text{ proceed } -> \text{ break } \\
od; \\
x++
\end{align*}
\]
Wait for another process to set *proceed* to true:

```plaintext
x = y;
do
:: proceed -> break;
od;
x++;
```

is equivalent to

```plaintext
x = y;
proceed; // expression – block until true.
x++;
```
Conditional Expressions

• Allows us to obtain a value that depends on the result of a boolean expression

Max = (a > b -> a:b)

• Note that conditional expressions must be contained within the parentheses
Repetitive Statements

• **do** statement is the only repetitive statement in Promela

• Starts with **do**, ends with **od**

• Completion of the sequence of statements causes the execution to return to the beginning of the do statement
Counting Loops

• There are no counting loops (e.g., for loops) in Promela

• You can implement them yourself

....
do
  :: i > N -> break
  :: else -> sum = sum+1;
  
  i++;
od;
Jump Statements

- **goto** statements cause control to jump to a label
  
  - Example
  
  ```
  do
  :: i > N -> goto exitloop
  :: else -> ....
  od;
  exitloop:
  ```
Jump Statements

- Jumps are allowed only within a process

- Jump statement labels need to be unique, per process

- Cannot place labels in front of guards
Data and Program Structures
Arrays

• Arrays are a sequence of data values that have the same type
  – Example

```c
active proctype P()
{
    int a[5];

    do
    ......
    od;
    printf ("The sum of the numbers is %d\n", sum)
}
```
Arrays

• Array index starts with 0

• Array bounds are constants and cannot be changed

• Promela only allows one-dimensional arrays
Type Definitions

- Can define compound types using `typedef`
  
  ```
  typedef Time
  {
    int hour, minute, second;
  }
  
  active proctype P()
  {
    Time T;
    T.hour=23; T.minute = 59; T.second = 32
  }
  ```
Inline

• Promela does not have functions or procedures

• You can group statements together using the **inline** construct

• Does not create new scope for local variables
typedef Time
{
    int minute, hour, second;
}

inline setTime (T, min, hr, sec){
    T.minute = min; T.hour = hr; T.second = sec
}

active proctype P(){
    Time t;
    setTime(t, 59,23,32)
}
Verification in SPIN

• Just like in Alloy, in SPIN assertions are used to verify the correctness of a model.

• **Assertions** are placed between program statements and are evaluated by the model checker for counterexamples.
Example

byte x = 7;

active proctype increment() {
    int oldx = x;

    printf("Hello, world\n") ;
    printf("x = %d\n", x) ;
    x++ ;
    printf("x = %d\n", x) ;
    assert( x >= oldx ) ;
}

active proctype P()
{
  int a = 5, b=5;
  int max;
  assert (a>=0 && b>=0); //pre-condition
  if
    :: a >= b -> max = a;
    :: b >= a -> max = b+1;
  fi;
  assert (a>=b -> max ==a : max == b) //post-condition
  ;
  printf ("The max of %d and %d = %d\n", a, b, max)
}