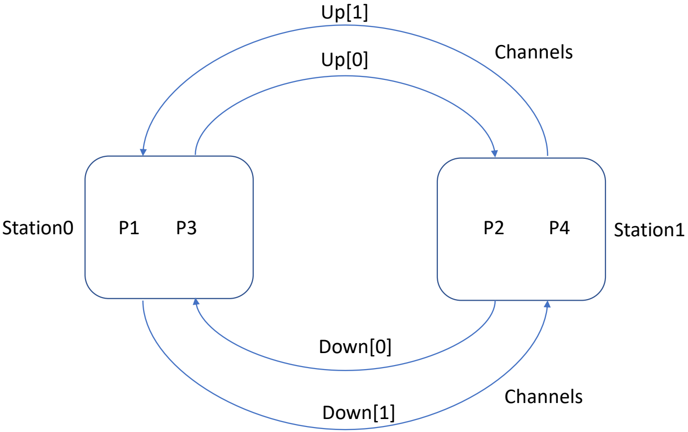
Introduction of Assignment 1

Zhen Dong, Abhik Roychoudhury

July 1, 2018

Table of Contents

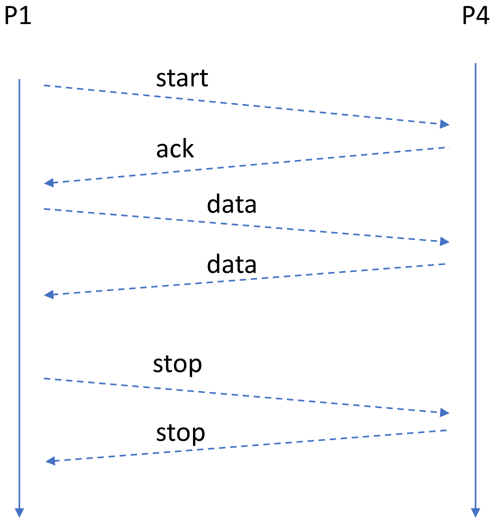
Question1: Communication between two stations (1)



Double channels communication protocol

Question1: Communication between two stations (2)

A communication demonstration:



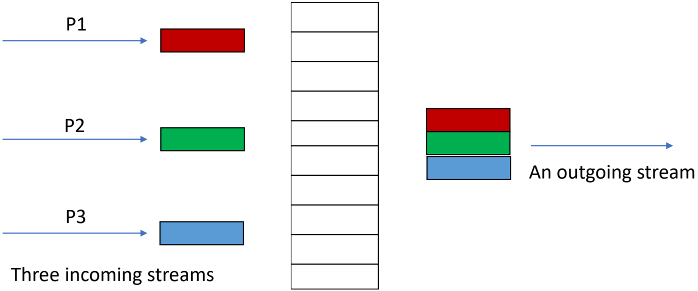
Question1: Communication between two stations (3)

Questions:

- Verification: wrirte LTL to verify the provided model such that each commuication between the two statioms is started with signal "start" and termined with "stop" **[3 marks]**,
- Augumentation: augment the provided model to make the above property true **[4 marks]**.

Question2: A deadlock verification (1)

Write a Promela model of a network node with three incoming streams of messages, and one outgoing stream.



A local buffer with fixed length

Question2:A deadlock verification (2)

Question:

- write a Promela model for reassembling incoming packets using a local buffer as a message and sending it out. Require that each message to be sent on the outgoing stream contains a structure of 3 fields, one field of type red, one of type green and of type blue **[2 marks]**,
- Use handshake for the incoming messages **[2 marks]**,
- Show how the reassembly deadlock can happen **[2 marks]**,
- Define the key property in LTL and prove it can be violated and show the counter-example **[2 marks]**.

Introducing Promela Language

Promela = Process Meta Language

- A specification language ! No programming language !
- Used for system description :
  - Specify an abstraction of the system, not the system itself.
- Emphasize on process synchronization & coordination, not on computation.
- Promela uses nondeterminism as an abstraction technique.
- Suitable for software modeling, not for hardware.

SPIN = Simple Promela Interpreter

- A simulator for Promela programs.
- And a verifier for the properties of Promela programs.
- In simulation mode, SPIN gives quick impressions of system behavior.
  - Nondeterminism in specification is “randomly solved”.
  - No infinite behaviors.
- In verification mode, SPIN generates a C program that constructs an implementation of the LTL model-checking algorithm for the given model.
  - Then one has to compile/run this C program to get the result.
  - ... which may provide a trace for the bugs in the model.

Hello world

- Promela program `hello.pml` :

```
active proctype main(){
   printf("Hello world")
}
```

- Simulating the program :

```
$ spin hello.pml
hello world
1 process created
```

- `proctype` = declares a new process type.
- `active` = instantiate one process of this type.

Producers/Consumers

```
mtype = { P,C }; /* symbols used */
mtype turn = P;  /* shared variable */

active proctype producer(){
   do
   :: (turn == P) ->   /* Guard */
            printf("Produce\n");
            turn = C
   od
}
active proctype consumer(){
again:
   if
   :: (turn == C) ->   /* Guard */
            printf("Consume\n");
            turn = P;
            goto again
   fi
}
```

Condition statements and nondeterminism

- Proctype consumer rewritten :

```
again:
    (turn == C);
    printf("Consume\n");
    turn = P;
    goto again;
```

  - Condition statement, blocking the process until the condition becomes true.

- Nondeterminism :

```
byte count;
active proctype counter(){
    do
    :: count++
    :: count--
    :: (count==0) -> break
    od
}
```

Atomic statements

- Promela focuses on modeling distributed systems.

```
byte a;
active proctype p1(){
    a=1;
    b=a+b
}
active proctype p2(){
    a=2;
}
```

- Atomicity needed for avoiding race conditions :

```
atomic{ a=1; b=a+b }
atomic{ tmp=y; y=x; x= tmp }
```

Data objects

- Data can only be global or process local.
- Integer data types + bits + boolean.
- C syntax for variable declarations.
- One-dimensional arrays only.
- `mtype` = list of symbolic values, range 1..255.
    - A single list for a Promela program !

      ```
      mtype = { A, B, C };
      mtype = { 1, 2, 3 }; /* union of the two sets */
      ```
- Record structures definable :

```
typedef Field{
   short f=3; byte g
}
typedef Record{
   byte a[3];
   Field fld;
}
```

- Can be used for defining multidimensional arrays.

Channels

- Variables modeling communication channels between processes.
- Must be declared globally, if needed by two distinct processes.

```
chan queue = [10] of { mtype, short, Field }
```

  - 10 message buffer, each message composed of 3 fields.

- Sending messages :

```
queue!expr1,expr2,expr3;
queue!expr1(expr2,expr3)
```

  - `expr1` used as message type indication.

- Receiving messages :

```
queue?var1,var2,var3;
queue?var1(var2,var3)
```

- Conditional reception :

```
queue?A(var2,var3);
queue?var1,100,var3
queue?eval(var1),100,var3
```

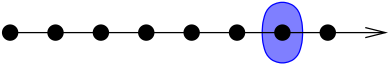
  - Execute only when first field matches value of `var1`.

Linear temporal logic

Introducing Liner Temporal Logic

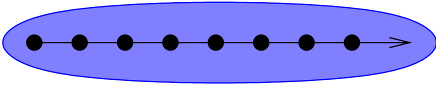
LTL specifications

finally p



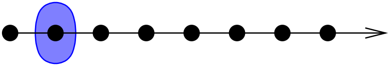
F p

globally p



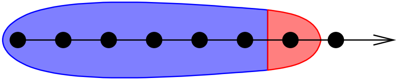
G p

next p



X p

p until q



p U q

LTL syntax with SPIN

- Grammar:
  - `ltl ::= opd | ( ltl ) | ltl binop ltl | unop ltl`
- Operands (`opd`):
  - `true`, `false`, and user-defined names starting with a lower-case letter
- Unary Operators (`unop`):
  - `[]` (the temporal operator always)
  - `<>` (the temporal operator eventually)
  - `!` (the boolean operator for negation)
- Binary Operators (`binop`):
  - `U` (the temporal operator strong until)
  - `V` (the dual of U, release): (p V q) means !(!p U !q))
  - `&&` (the boolean operator for logical and)
  - `||` (the boolean operator for logical or)
  - `->` (the boolean operator for logical implication)
  - `<->` (the boolean operator for logical equivalence)

LTL model checking: intuition

To model check if $M \models \phi$, SPIN does

- build an automaton $A_{\neg\phi}$ that encodes all violations of ϕ,
- consider the synchronous execution of M and $A_{\neg\phi}$
  $\Longrightarrow A_M \times A_{\neg\phi}$ represents the paths in M that do not satisfy ϕ.

$A_{\neg\phi}$ ("never claim") can be seen as a monitoring machine that accepts some infinite executions of the system. If there exists an execution accepted by $A_{\neg\phi}$, that execution is a violation of ϕ.

Verifying LTL properties with SPIN 1/2

- Suppose we want to verify that a system satisfies a property. Example: in the system `foo.pml`, a boolean variable `b` is always true.
- Write the corresponding LTL formula using some simple symbols as atomic propositions (usually, single characters): `[] p`.
- Write the symbol definitions:
  `> echo ''#define p (b==true)'' > foo.aut`
- Generate the never claim corresponding to the negation of the property:
  `> spin -f '!([] p)' >> foo.aut`

Verifying LTL properties with SPIN 2/2

- Generate the verifier:
  `> spin -a -N foo.aut foo.pml`
- Option `-N file.aut` adds the never claim stored in `file.aut`
- Compile and run the verifier:
  `> gcc -o pan pan.c`
  `> ./pan -a`
- When a never claim is present and `-a` option is used, the verifier reports the existence of an execution accepted by the never claim. This execution corresponds to a violation of the property.

Remote references

- Typically, in order to test the local control state of active processes, we use the remote reference `procname[pid]@label`.
- This function return a non-zero value iff the process `procname[pid]` is currently in the local control state marked by `label`.
- Example:

  ```
  []!(mutex[0]@critical && mutex[1]@critical)
  ```

- We can also refer to the current value of local variable by using `procname[pid]:var`

Predefined global variables and functions

- The predefined local variable `_pid` stores the process instantiation number (pid) of a process.
- The predefined global variable `_last` stores the pid of the process that performed the last execution.
- The function `enabled(pid)` returns true if the process with identifier `pid` has at least one executable statement in its current control state.

Useful links

- Spin download: http://spinroot.com/spin/Man/README.html
- Spin tutorial: http://spinroot.com/spin/Man/
- Promela grammar: http://spinroot.com/spin/Man/promela.html
- Basic manual: http://spinroot.com/spin/Man/Manual.html
- Spin verify LTL properties:
  http://disi.unitn.it/ agiordani/fm/L4/main.pdf