COMP30112: Concurrency

Topics 5.2: Properties

Howard Barringer

Room KB2.20: email: Howard.Barringer@manchester.ac.uk

April 2009
Outline

Topic 5.2: Properties

Properties in general
Specifying Safety Properties in FSP
Example - Semaphores and Mutual Exclusion
Example - Single Lane Bridge - FSP Model
Example - SLB - Java Implementation
Fairness
Outline

Topic 5.2: Properties

Properties in general
Specifying Safety Properties in FSP
Example - Semaphores and Mutual Exclusion
Example - Single Lane Bridge - FSP Model
Example - SLB - Java Implementation
Fairness
Safety and Liveness Properties

- **Safety**: property holds in *all* states — nothing bad
Safety and Liveness Properties

- **Safety**: property holds in *all* states — nothing bad
- **Liveness**: property *eventually* holds — something good
Safety and Liveness Properties

- **Safety**: property holds in *all* states — nothing bad
- **Liveness**: property *eventually* holds — something good

Examples:
Safety and Liveness Properties

- **Safety**: property holds in *all* states — nothing bad
- **Liveness**: property *eventually* holds — something good

Examples:

- Safety:
  - Deadlock-freedom
  - Mutual exclusion
Safety and Liveness Properties

- **Safety**: property holds in *all* states — nothing bad
- **Liveness**: property *eventually* holds — something good

**Examples:**

- **Safety**:
  - Deadlock-freedom
  - Mutual exclusion

- **Liveness**:
  - a result!
  - fairness
  - restrict to *progress*
Outline

Topic 5.2: Properties

Properties in general

Specifying Safety Properties in FSP

Example - Semaphores and Mutual Exclusion
Example - Single Lane Bridge - FSP Model
Example - SLB - Java Implementation

Fairness
A Faulty Traffic Light Example

Lights = (red -> redamber -> (green -> amber -> Lights
        | redambergreen -> Loop)),
Loop = (red -> Loop).
Cycle = (red -> green -> Cycle).
||System = (Lights || Cycle).

★ What’s the LTS for System? ★
Specifying Cycle as a Safety Property

property
PCycle = (red → green → PCycle).
And then when composed ...

\[
\text{Lights} = (\text{red} \rightarrow \text{redamber} \rightarrow (\text{green} \rightarrow \text{amber} \rightarrow \text{Lights} \\
| \text{redambergreen} \rightarrow \text{Loop})),
\]

\[
\text{Loop} = (\text{red} \rightarrow \text{Loop}).
\]

property
\[
\text{PCycle} = (\text{red} \rightarrow \text{green} \rightarrow \text{PCycle}).
\]

\[
||\text{System} = (\text{Lights} || \text{PCycle}).
\]

The composition yields a property violation - there is a loop that has red not followed by a green action.
The composite LTS showing property violation
What happens here ...

Lights = ( red -> redamber -> ( green -> amber -> Lights
| redambergreen -> Loop ) ),

Loop = (red -> Loop).

Alt = (red -> green -> Alt).

property
PCycle = (red -> green -> PCycle).

||System = (Lights || Alt || PCycle).
Definition of Safety Property in FSP

Safety property $P$ defines a deterministic process that asserts that any trace including actions in the alphabet of $P$ is accepted by $P$.

Finding LTS for property $P$:

- Define *State Alphabet*, for state $s$:
  \[ \alpha(s) = \{ a | \exists t : (s \xrightarrow{a} t) \in \sigma \} \]

- Find $\text{lts}(P)$
- Form $\text{lts}_{\text{prop}}(P)$: add transitions
  \[ \{(s \xrightarrow{a} \text{ERROR}) | s \in S, a \in \alpha(P), a \notin \alpha(s)\} \]

Now compose $\text{lts}_{\text{prop}}(P)$ with $\text{lts}(T)$ for target process $T$. 
**Transparency**: Property must not change behaviour of a process with correct behaviour.

Properties must therefore be deterministic.

**Specifying that an Action never occurs**: Simply add to alphabet of property:

\[
\text{property PROP1} = \text{STOP} + \{ \text{never} \}.
\]

\[
\text{property PROP2} = (\text{red} \rightarrow \text{green} \rightarrow \text{PROP2}) + \{ \text{redambergreen} \}.
\]
Outline

Topic 5.2: Properties

- Properties in general
- Specifying Safety Properties in FSP
- Example - Semaphores and Mutual Exclusion
- Example - Single Lane Bridge - FSP Model
- Example - SLB - Java Implementation
- Fairness
Semaphores

Introduced by Dijkstra for inter-process synchronisation.

- **Semaphore** $s$ is a non-negative integer variable.
- Once initialised, only two operations allowed
  - $down(s)$ — when $s > 0$ do decrement $s$
  - $up(s)$ — increment $s$
Semaphores

Introduced by Dijkstra for inter-process synchronisation.

- **Semaphore** $s$ is a *non-negative* integer variable.
- Once initialised, only two operations allowed
  - $down(s)$ — *when* $s > 0$ do decrement $s$
  - $up(s)$ — increment $s$

Semaphores are passive objects. Thus, model a semaphore in Java as a monitor class. $down(s)$ requires condition synchronisation.
const Max = 3
range Int = 0..Max

SEMAPHORE(N=0) = SEMA[N],
SEMA[v:Int] = ( up -> SEMA[v+1]
    | when (v>0) down -> SEMA[v-1] ),
SEMA[Max+1] = ERROR.
Mutual Exclusion Example

LOOP = (mutex.down ->
        enter -> exit ->
        mutex.up -> LOOP).

|| SEMADEMO = ( p[1..3]:LOOP
              || {p[1..3]}::mutex:SEMAPHORE(1) ).

property
MUTEX = ( p[i:1..3].enter -> p[i].exit -> MUTEX).

|| CHECK = ( SEMADEMO || MUTEX ).
MUTEX fails

If SEMAPHORE is initialised to 2.

Trace to property violation in MUTEX:
  p.1.mutex.down
  p.1.enter
  p.2.mutex.down
  p.2.enter
Topic 5.2: Properties

Outline

Topic 5.2: Properties

- Properties in general
- Specifying Safety Properties in FSP
- Example - Semaphores and Mutual Exclusion
- Example - Single Lane Bridge - FSP Model
- Example - SLB - Java Implementation
- Fairness
Single Lane Bridge — No Crashes Please!
Single Lane Bridge Model

\[ \text{CAR} = (\text{enter} \rightarrow \text{exit} \rightarrow \text{CAR}). \]

\[ \text{NOPASS1} = C[1], \]
\[ C[i:\text{ID}] = ([i].\text{enter} \rightarrow C[i\%N+1]). \]

\[ \text{NOPASS2} = C[1], \]
\[ C[i:\text{ID}] = ([i].\text{exit} \rightarrow C[i\%N+1]). \]

\[ ||\text{CONVOY} = ([\text{ID}]:\text{CAR} \mid\mid \text{NOPASS1} \mid\mid \text{NOPASS2}). \]

\[ ||\text{CARS} = (\text{red}:\text{CONVOY} \mid\mid \text{blue}:\text{CONVOY}). \]
BRIDGE = BRIDGE[0][0],

BRIDGE[nr:T][nb:T] =
    ( when (nb==0) red[ID].enter -> BRIDGE[nr+1][nb]
    | red[ID].exit   -> BRIDGE[nr-1][nb]
    | when (nr==0) blue[ID].enter -> BRIDGE[nr][nb+1]
    | blue[ID].exit  -> BRIDGE[nr][nb-1]
    ).
property ONEWAY = ( red[ID].enter -> RED[1] 
   | blue[ID].enter -> BLUE[1] ),

RED[i:ID] = ( red[ID].enter -> RED[i+1] 
   | when (i==1) red[ID].exit   -> ONEWAY 
   | when (i>1)  red[ID].exit   -> RED[i-1] ),

BLUE[i:ID] = ( blue[ID].enter -> BLUE[i+1] 
   | when (i==1) blue[ID].exit   -> ONEWAY 
   | when (i>1)  blue[ID].exit   -> BLUE[i-1] ),

||SingleLaneBridge = (CARS || BRIDGE || ONEWAY ).
Topic 5.2: Properties

Outline

Topic 5.2: Properties

Properties in general
Specifying Safety Properties in FSP
Example - Semaphores and Mutual Exclusion
Example - Single Lane Bridge - FSP Model
Example - SLB - Java Implementation
Fairness
class SafeBridge extends Bridge {
    private int nred = 0; private int nblue = 0;

    synchronized void redEnter() throws InterruptedException {
        while (nblue>0) wait();
        ++nred;
    }

    synchronized void redExit(){
        --nred;
        if (nred==0) notifyAll();
    }

    synchronized void blueEnter() throws InterruptedException {...}
    synchronized void blueExit(){...}
}
Outline

Topic 5.2: Properties

Properties in general
Specifying Safety Properties in FSP
Example - Semaphores and Mutual Exclusion
Example - Single Lane Bridge - FSP Model
Example - SLB - Java Implementation
Fairness
 topic 5.2: properties

fairness

• unconditional: all unguarded actions will eventually be selected
• weak: all actions whose guard becomes continuously true will eventually be selected
• strong: all actions whose guard becomes true infinitely often will be infinitely often executed
Example - Fairness Required?

VAR = VAR[0],

VAR[x:0..1] =
  ( when (x == 0) settrue -> VAR[1]
    | when (x == 1) setfalse -> VAR[0]
  ).

TRUE = (settrue -> TRUE)+{setfalse}.
FALSE = (setfalse -> FALSE)+{settrue}.

||SYSTEM = ({t1,t2}::FALSE || s:TRUE || {t1,t2,s}::VAR).
class Var {
    boolean x = true;
    
synchronized void setfalse(String id) throws InterruptedException {
        while (x==false) { wait(); }
        x=false;
        notify();
    }

    synchronized void settrue(String id) throws InterruptedException {
        while (x==true) { wait(); }
        x=true;
        notify();
    }
}
class False extends Thread {
    String id;  Var x;

    False(String i, Var y) {id = i; x = y;}

    public void run(){
        while (true) {
            try {
                x.setfalse(id);
            } catch (InterruptedException e) {}
        }
    }
}
class True extends Thread {...}

class Life {
    public static void main (String [] args) {
        Var x = new Var();
        False t1 = new False("T1",x);
        False t2 = new False("T2",x);
        True s = new True("S",x);
        t1.start(); t2.start(); s.start();
    }
}