COMP31212: Concurrency

Topics 5.2: Properties
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
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 Washing Machine Example

\[ \text{CYCLE} = (\text{wash} \rightarrow \text{rinse} \rightarrow \text{dry} \rightarrow \text{CYCLE}). \]

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

property CYCLEPROPERTY =
    (wash -> rinse -> dry -> CYCLEPROPERTY).

LTS:
Washing machine model

range $S = 0..3$

WASHER = (on -> STATE[0] ),
STATE[s:S] =
( when (s==0) closedoor -> STATE[s+1]
| when (s==0) opendoor -> buzz -> STATE[0]
| when (s==1) wash -> STATE[s+1]
| when (s==2) pause -> PAUSED[s]
| when (s==2) rinse -> STATE[s+1]
| when (s==3) pause -> PAUSED[s]
| when (s==3) dry -> off -> WASHER),


Let’s check this by composition with the cycle property.
What is the LTS of the following parallel composition of a process with a property?

\[ \text{||TEST} = (\text{WASHER} \mid\mid \text{CYCLE}_{-}\text{PROPERTY}). \]

Answer:
Detecting safety property violations automatically

The existence of a path from the initial state to the error state (-1) means that the WASHER process does NOT satisfy the cycle property.

There is a trace of the WASHER process which does not follow the sequence required by the cycle property.

So to check whether a process satisfies a safety property:

(1) Compose the process in parallel with the safety property,
(2) Check whether the composite has paths to the error state or not.
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 \( \text{lts}_{prop}(P) \): add transitions
  \[
  \{(s \xrightarrow{a} \text{ERROR}) | s \in S, a \in \alpha(P), a \not\in \alpha(s)\}
  \]

Now compose \( \text{lts}_{prop}(P) \) 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:

```plaintext
property PROP1 = STOP + \{ never \}.

property PROP2 = (wash -> rinse -> dry -> PROP2) + \{ iron \}.
```
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

\[
\text{LOOP} = (\text{mutex.down} \rightarrow \\
\quad \text{enter} \rightarrow \text{exit} \rightarrow \\
\quad \text{mutex.up} \rightarrow \text{LOOP}).
\]

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

property
\[
\text{MUTEX} = ( p[i:1..3].\text{enter} \rightarrow p[i].\text{exit} \rightarrow \text{MUTEX}).
\]

\[
\text{|| CHECK} = ( \text{SEMADEMO} \text{|| MUTEX}).
\]
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
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

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

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

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

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

\[ ||\text{CARS} = (\text{red}:\text{CONVOY} || \text{blue}:\text{CONVOY}). \]
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 $\rightarrow$ RED[1] \\
| blue[ID].enter $\rightarrow$ BLUE[1] ),

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

BLUE[i:ID] = ( blue[ID].enter $\rightarrow$ BLUE[i+1] \\
| when (i==1) blue[ID].exit $\rightarrow$ ONEWAY \\
| when (i>1 ) blue[ID].exit $\rightarrow$ 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
Single Lane Bridge — Java aspects

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
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();
   }
}